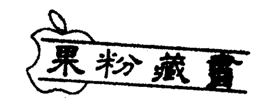
6502
微处理机及其应用

北京师范大学出版社

6502微处理机及其应用

荣树熙　张开敬　　编

北京师范大学出版社



内 容 简 介

本书对 ROCKWELL 6502微处理机及其部分接口芯片进行了系统、详尽地论述。全书分五章，第一、二章介绍了6502 的寻址方式和指令系统，6502汇编语言的程序设计方法；第三、四章介绍了以6502为CPU的典型计算机，小汇编和编辑/汇编程序的使用，6500系列的一些常用的可编程接口芯片；最后一章介绍了一些应用方面的课题。

本书可供大专院校师生，工程技术人员和从事计算机教学的中学教师参阅。

前　言

本书是根据作者在北京师范大学无线电电子学系讲授6502微处理机课程的讲义而编撰，其中，对ROCKWELL6502微处理机及其部分接口芯片进行了比较详尽、系统地论述，在最后一章还专门讨论了它的部分典型的实际应用。

全书共分五章，首先从6502的内部结构及工作时序开始，然后详细讨论了其中十三种寻址方式和五十六条指令。接着分析了大量的例题，说明6502汇编语言的编程方法。第三章对以6502为CPU的三种典型的计算机，即SYM-1、AIM-65和APPLE Ⅱ 进行了概括的介绍，并对APPLE Ⅱ 和AIM-65的监控程序、小汇编和编辑/汇编程序的使用进行了详细的说明。第四章主要叙述6500系列一些常用的可编程接口芯片。为使读者易于理解，我们列举了许多编程实例来说明芯片的各种功能。最后一章专门介绍了一些应用的课题，它们一方面可以帮助读者进一步掌握前面所学的知识；另一方面这些课题在实际工作里的应用会对读者有所帮助。本书所列举的大量程序均已在计算机上进行了验证。

此书编写过程中，北京师范大学无线电电子学系李立文同志协助做了许多工作，并对其中部分程序进行了上机试验，在此谨表谢意。

由于作者水平有限，错误缺点在所难免，敬请广大读者和计算机工作者批评指正。

作　者

Retro Workshop
果粉工作室

目　　录

6502微处理机概述

美国ROCKWELL公司推出的6502微处理机是一种广泛应用的大规模集成电路芯片。它同已经为我国普遍了解的INTEL8080/8085、ZILOG Z80和MOTOROLA6800一样，都是功能良好的8bit微处理机。以它为基础的微型计算机系统如 APPLE、ATARI、BBC（英国）等在全世界享有盛名，特别是在教育领域它们是首屈一指的。6502为基础的单板机在美国也很有声誉。其中AIM-65和 SYM-1 单板机在美国被认为是性能价格比最好的一类单板机，在美国的大学里应用相当普遍。6502也被应用于工业控制、仪表、空间技术等许多方面。本书的目的就是较全面地介绍6502及其应用。

6502微处理机的内部结构

6502～6515是一系列的由ROCKWELL生产的8bit 微处理机，其中以6502功能为最强，应用最为广泛。我们这里仅仅介绍6502，别的芯片读者可参阅有关资料手册。

图1是6502的内部结构框图。由图可以看出，它有八根数据线和十六根地址线，所以数据总线宽度为 8bit，地址总线十六根，所以寻址空间为65536个单元。这些同其它八位微处理机是没有差别的。它大体上可以分为两部分，即寄存器部分和控制部分。寄存器部分除了程序计数器是十六位以外，其余均为八位，这同各种微处理机比较起来是最少的。一般看来，好象内部寄存器少，功能就一定不如包含内部寄存器多的微处理机，其实不然，6502

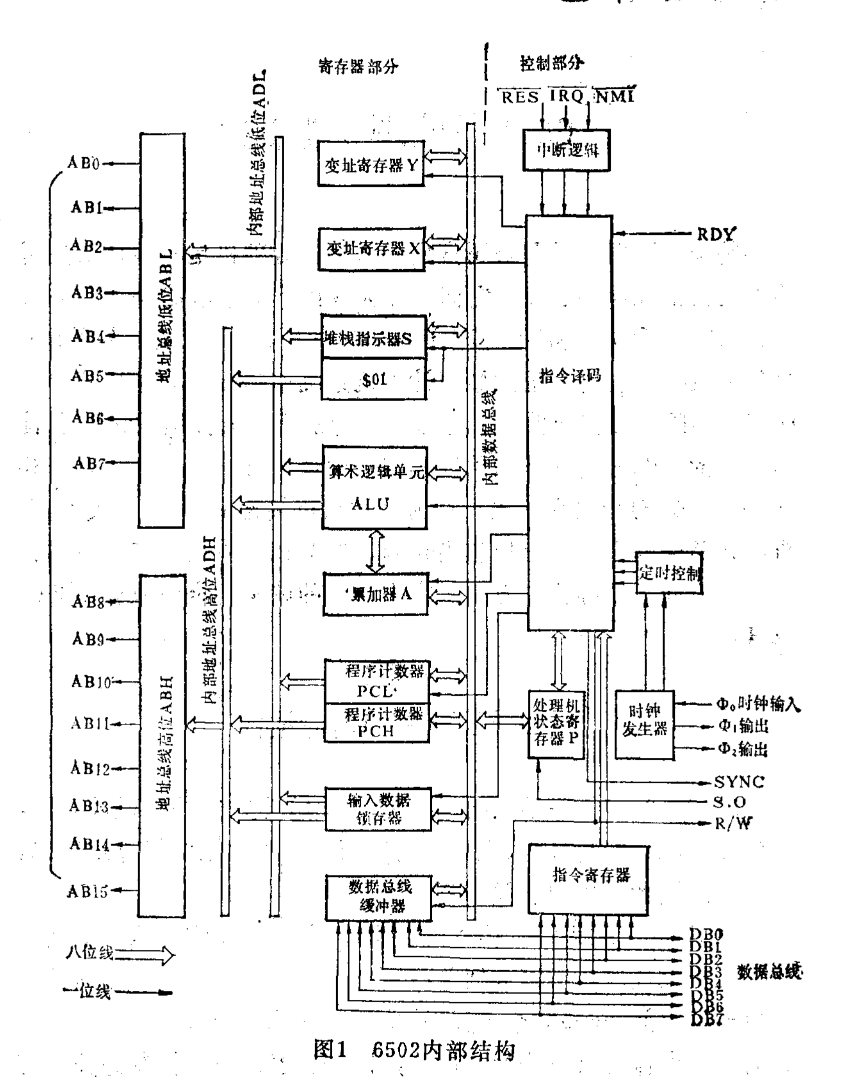


图1　6502内部结构

所具有的灵活多样的寻址方式，(共十三种，以后详述)对此完全可予以补偿。6502采用零页寻址方式的指令很多，这些指令占存贮空间少，执行速度快。因此零页的256个存贮单元就好象成了它

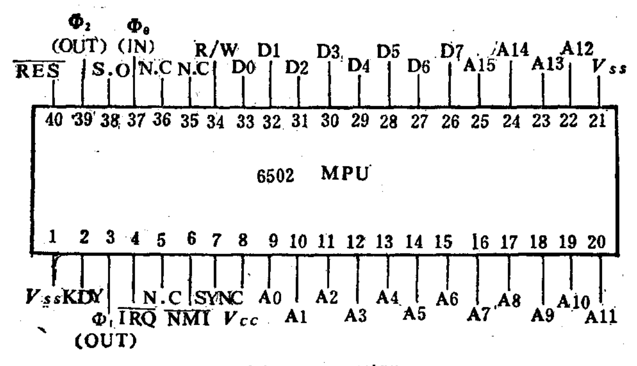


图2 6502引腿

的内部寄存器一样。由于变址寄存器X、Y和堆栈指示器(指可编程部分)均是八位，所以在执行同它们有关的指令时，就有利于加快速度，这一点同Z80、6800、8085等相比是有较明显的好处的。当然由于堆栈指示器只有八位可以编程，因此使用6502构成的计算机的堆栈只能固定放在存贮器第1页。6502内部包括了时钟振荡电路，如果在 ϕ_0 和 ϕ_2（37、39引腿）接上电阻电容，即可与内部振荡电路构成时钟发生器，不过一般我们为了使时钟频率稳定都用晶体振荡器作为时钟，这时 ϕ_0 引腿应输入一个频率为1兆赫的晶体振荡信号。ϕ_1、ϕ_2 是6502的两个输出端，提供两个基本的时钟信号，以供存贮器及外设接口使用。ALU算术逻辑单元是完成算术及逻辑运算的基本部件，运算是八个二进制位并行进行的。它的输入一个是累加器A，另一个是从存贮器经由数据总线来的八位二进制数，算出的结果又送回到累加器A中。由此可以看出累加器A在运算中十分重要，它既是数据的来源地，又是算出的结果所要送到的目的地，累加器A还可以通过编程实现左右移位和循环移位。变址寄存器X、Y在实现变址寻址方式中有很大的作用，同时它们也可完成加1、减1以及比较等简单的运算。

处理机状态寄存器又称为标志寄存器，它的各个标志位由指令执行的结果所决定，它们是实现条件跳转的依据，在程序设计中占有很重要的地位。有关各个标志位的作用，我们在下一章详细说明。程序计数器由PCL和PCH组成，一共十六位，前者是低八位，后者是高八位。以下我们来一一介绍6502各条引线的功能。

数据总线(D_0～D_7) 它是6502与外界进行信息交换的通道，一共八条线。它可以工作在输入方式又可以工作在输出方式，在一个时间内，它究竟工作在哪一种方式则由R/W线决定。数据线每条线至少能驱动130PF电容和一个标准TTL负载。每条线都是三态的双向线。

地址总线(AB0～AB15) 一共十六根输出线，每条线至少能驱动130PF电容和一个标准TTL负载。6502在工作过程中，总是先将程序计数器(PCH、PCL)的内容送到地址总线，并到由其决定的存贮单元中取出操作码，通过数据总线送到指令寄存器，指令译码器根据操作码的性质发出所需要的各种控制信号，程序计数器自动加1并决定下一步的操作。

读/写线(R/W) 这是6502的一根输出线。正是由它决定着数据总线上信息传输的方向。当它为1（高电平)时，信息由外部输入6502内部，即数据总线为输入方式工作——这个过程我们称之为读操作，当它为0（低电平)时，信息传送方向是由6502向外部，即数据总线为输出方式工作——这个过程我们称之为写操作。它的带载能力与地址线相同。

准备好线(RDY) 这是一根从外部向6502的输入线。它的作用是可以延长执行周期。当它变为低电平(0)时，6502的读操作周期将延续下去——即R/W控制线和地址线（AB0～AB15）上的电平将持续不变，一直到RDY线电平变高（1)为止。但特别要注意RDY线的这种延续作用仅对读操作有效。

不可屏蔽中断请求（$\overline{\text{NMI}}$）线 这是一条由外部对6502的输

入线。如果这条线上出现了由1电平变为0电平的负跳变，处理器在完成了正在执行的那一条指令之后，即转入响应中断的操作。由于它不受中断屏蔽标志位的影响，所以称这种响应为不可屏蔽中断响应。注意，处理器只对$\overline{\text{NMI}}$线上的负跳沿进行识别和响应，而不象$\overline{\text{IRQ}}$线那样是对低电平进行识别和响应。

可屏蔽中断请求($\overline{\text{IRQ}}$)线： 这是一条由外部对6502的输入线。如果这条线上出现低(0)电平时，只要处理机状态寄存器(P)中的中断屏蔽标志位的内容为0，处理机在完成正在执行的指令之后，即转入中断响应操作。但是如果中断屏蔽标志位为1，即使$\overline{\text{IRQ}}$线为低电平，处理机亦不可能响应。因此我们把上述响应称之为可屏蔽中断响应。注意，处理机是对$\overline{\text{IRQ}}$线上的低电平进行识别和响应(如果可能的话)，而不是象$\overline{\text{NMI}}$那样仅对负跳沿进行识别和响应。因此要求$\overline{\text{IRQ}}$线上的低电平要保持足够长的时间，即至少要保持到本条指令执行完了。在响应周期中，处理机自动将程序计数器以及处理机状态寄存器的内容送往堆栈保存，并将中断屏蔽标志位置成1状态。中断服务程序中应安排清除中断输入的指令。

有关不可屏蔽中断以及可屏蔽中断在以后的内容里还将专门加以叙述。

复位($\overline{\text{RES}}$)线 这是一条输入线。我们正是用这条线为低电平来强迫处理机进入初始化。在加电时，应使这条线保持低电平，当+5V电源以及晶振稳定之后，这条线变成高电平。从这时开始处理机经过六个时钟周期首先从指定的存贮单元FFFC(十六进制)取出数据送程序计数器PCL；然后又从存贮单元FFFD(十六进制)取出数据送PCH。六个时钟周期之后，处理机将从PCH和PCL新内容所指向的地址开始运行。

同步信号(SYNC)线 这是一根输出线。当处理机工作在取操作码周期，这根线将变成高(1)电平。因此可以由它的信号来识

别处理机的工作周期。它保持高电平的时间为取操作码的整个时钟周期。

置溢出位(S·O)线　这是一条输入线。可以通过这一条线 用负跳沿使6502内部的处理机状态寄存器(P)的溢出标志位 (V) 置1。一般情况下这条线并不使用。

ϕ_0以及ϕ_1和ϕ_2线前面已经讲过，这里不再重复了。

电源(V_{cc}和V_{ss})线　V_{cc}接+5V，V_{ss}接0V(地)。电源变动范围是±5%，V_{cc}的极限值为7V。

图2中标有N.C字样的线，都是没有使用的空引腿。

6502的时序

我们来谈谈6502微处理机的工作时序。微处理机的重要组成部分之一是它的时钟，时钟对于它就象心脏对于人一样重要。人体各部 分 工 作 都离不了以平稳节拍跳动的心脏，微处理机正常工作首先要求有一个周期稳定的时钟信号。对6502来讲，这个时钟一般为1兆赫(10^6赫)左右。它一般由一个晶体振荡器提供，如图2晶体振荡器应接在ϕ_0端。而由微处理机组成的计算机系统都是按6502所提供的ϕ_1和ϕ_2作为系统时钟来工作的。其实ϕ_2 同 ϕ_0 同相，ϕ_1和ϕ_2反相，6502内部不过是加了两级反向器以提供有足够驱动能力的信号。图3画出了ϕ_1和ϕ_2的波形图。由于考虑到 ϕ_1、ϕ_2之间的延时以及可能允许的变化，所以图 3 中的波形看上去完全不象两个互为反相的信号。在实际的计算机中6502的ϕ_1 和 ϕ_2是互为反相的。图中 T_D：延迟时间，T_F：下降时 间，T_R：上 升时间。PWHϕ_1：时钟ϕ_1脉冲高电平宽度，PWHϕ_2：时钟ϕ_2脉冲高电平宽度。T_{CYC}：时钟周期。ROCKWELL公司所提供的资料指出：对于时钟ϕ_1和ϕ_2脉冲高电平宽度所允许的最小值为430ns(毫微秒)；上升时间和下降时间最大为25ns；时钟周期最小为 1μs(微

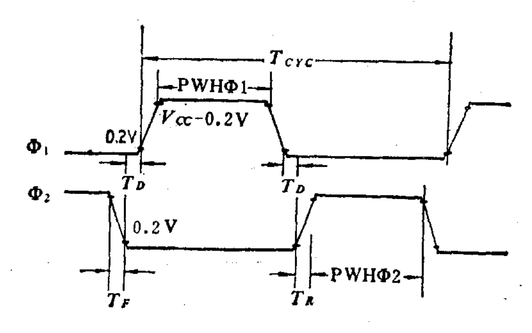


图3　6502两相时钟脉冲

秒)；延迟时间最小为0(即ϕ_1同ϕ_2反相)。以后为了方便起见，我们把ϕ_1的高电平时间称作ϕ_1期间，而把ϕ_2为高电平时间则统称为ϕ_2期间。6502每一个时钟周期可同存贮器或外设接口交换一次信息，统一规定在ϕ_2期间交换信息，而在ϕ_1期间开始就提供新的地址和控制信号；在ϕ_1期间6502数据线处于高阻状态。大家知道，Z80和 8085每同存贮器交换一次信息的时间我们常称为一个机器周期，而一个机器周期又包含若干个时钟周期。现在我们注意到，对6502来讲，每一个时钟周期即可同存贮器或外设接口交换一次信息，因此对6502不必引用机器周期的概念。

图4和图5表示了6502微处理机从存贮器或外设读数据和写数据的详细时序。现说明如下：

由图4可以看出，从6502提供的读写(R/W) 控制线有一定的建立时间 (T_{RWS})，它的典型值是 100ns，最大值可能达 300ns。6502提供的地址也需要一定时间才能稳定，这就是地址建立时间(T_{ADS})，其典型值是200ns，最大可达300ns。为了保证6502读入的数据是可靠的，数据线上的数据至少需要稳定100ns(从ϕ_2的下降沿往前算）这就是图中的T_{DSU}——数据稳定时间。由此可以算出存贮器或外设接口的读出时间：

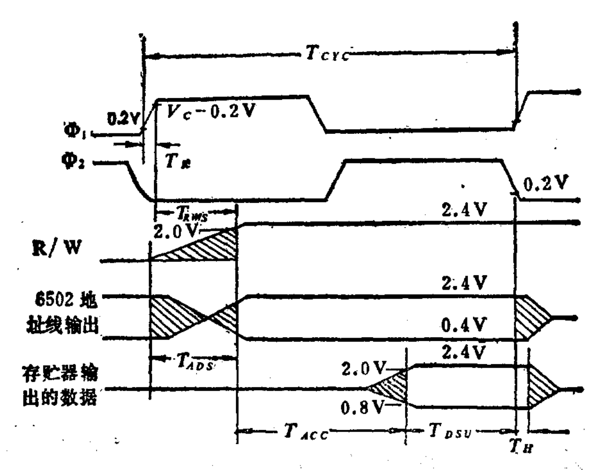


图4　6502从存贮器或外设读数据时序

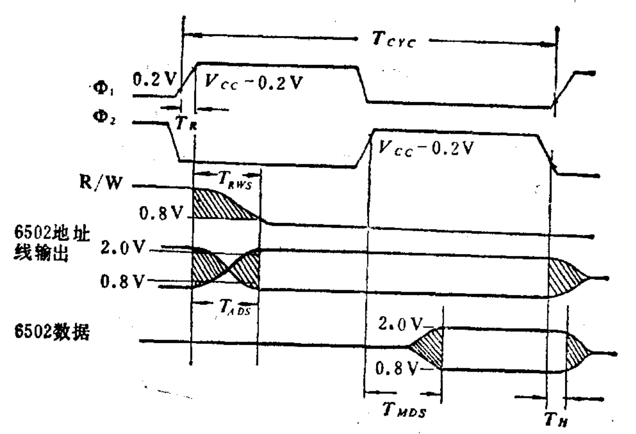


图5　6502向存贮器或外设写数据

$$
\begin{aligned}
T_R &= T_{CYC} - (T_{ADS} + T_{DSU} + T_R) \\
&= 1000 - (300 + 100 + 25) \\
&= 575(\mathrm{ns})
\end{aligned}
$$

因此，存贮器的读出时间不能长于575ns。这对存贮器目前的水平来讲是一般都可以达到的。因此对6502来讲，选取存贮器是比较容

易的。

图5是6502向存贮器或外设写入数据的时序图。在ϕ_1期间，读写(R/W)控制线以及地址线开始提供控制信号和地址，它们的建立时间最长可达300ns。6502在ϕ_2期间输出数据，这些数据在数据线上也要经过一定时间之后才能稳定，这就是图上表示的数据建立时间(T_{MDS})，长达150～200ns。

图4和图5都用到符号T_H，它称为数据保持时间。由图可以看出在ϕ_2结束之后，数据依然可以在数据线上保持一段很短的时间，这就是T_H，一般它的值是10～30ns。

我们将上面一系列的数据用表概括如下：

名　　称	符　号	最小值	典型值	最大值	单　位
时钟周期	T_{CYC}	1.0			μs(微秒)
时钟脉冲宽度 (以$V_{cc}-0.2V$测量)	PWHϕ1 PWHϕ2	430 430			ns(毫微秒)
上升和下降时间	T_F, T_R	—	—	25	ns
时钟ϕ_1和ϕ_2之间的延迟时间	T_D	0			ns
读写建立时间	T_{RWS}		100	300	ns
地址建立时间	T_{ADS}		200	300	ns
存贮器读出时间$T_{CYC}-(T_{ADS}+T_{DSU}+T_R)$	T_{ACC}	—	—	575	ns
数据稳定时间	T_{DSU}	100			ns
6502输出数据建立时间	T_{MDS}		150	200	ns

读写是6502的基本操作，任何指令的执行都离不了这两种操作。因此图4和图5即为6502最基本的时序图。

第一章　MPU6502的寻址方式及指令系统

本章讨论的内容是 6502 微处理机的指令寻址方式及指令系统。微型机指令的内容一般是由两个部分构成：第一部分为操作码，用来表示指令所规定的操作性质，如取数、存数、算术运算、转移操作等。第二部分为操作数，用来指出操作数在什么地方，所以又称这部分为地址码部分。如何得到操作数地址，这就是指令的寻址方式。一种微处理机所具有的指令的集合就称为这种微处理机的指令系统。

6502微处理机具有简捷明瞭的指令系统，灵活多样的寻址方式，使用起来很方便。在详细介绍MPU 6502的寻址方式和指令系统之前，我们首先来分析一下6502指令系统的特点。

1．指令条数少，指令格式整齐，易于掌握和记忆。

6502共有56条指令，十三种寻址方式。每条指令可对应有不同的寻址方式，所以按指令机器码区分又可认为6502有151条指令。

指令长度为1～3字节，其中第一字节一律为操作码，它决定6502完成某一种运算或操作。操作数或操作数地址码跟随在操作码之后占用一个或两个字节，由采用的寻址方式而定。

2．寻址方式灵活多样，便于提高编制程序的效率。

在各种八位微处理机中，6502的寻址方式是比较多的，共有十三种，而且寻址的灵活性好，可以方便灵活的在64K 存贮器的寻址范围内找到所需要的操作数。

3．广泛使用了零页寻址。

6502微处理机内部可供程序员访问的寄存器比较少，这是它的不足之处，但是由于有＄0000～＄00FF 这256个零页地址可供

使用零页寻址方式，而使这个不足之处得到了弥补。零页寻址方式的指令占用的字节较少，指令执行时间较短，这在节省内存和缩短程序执行时间两个方面为编制程序也提供了有利条件。

4. 和外围设备交换数据时采用存贮器映象方式。

微处理机和外围设备交换信息有两种方式：一种是存贮器映象方式，在这种方式中微处理机不设置专门的输入输出指令，而是把每一个外设当做存贮器的一个单元来处理，外设的地址码和存贮器的地址码都在64K 的范围内统一分配。从输入设备输入一个数据就相当一次读存贮器的操作。向输出设备输出一个数据，就相当于一次写存贮器的操作。对于存贮器映象方式，凡对存贮器可以使用的指令，都可以用于外设。

另一种是端口映象方式，在这种方式中，微处理机要设置专门的输入输出指令。在输入输出指令中对外设的寻址要另外设置外设地址码（即端口地址码），而不是和存贮器统一编址。

MPU 6502 和外设交换信息采用的是存贮器映象方式，所以对外围设备操作时，可供使用的指令就很多。例如传送，算术逻辑运算、移位等等，这比起端口映象方式要方便和灵活，但是由于外设和存贮器统一编址，所以外设占用了存贮器的地址，而使存贮器容量减小。

5. 标志寄存器各标志位设计合理。

与其它一些 8 位微处理机相比较，6502的标志位设计合理，能与每条指令操作密切配合，而给设计分支程序带来很大方便。

由于以上特点，而使 MPU 6502享有很好的声誉，拥有广泛的市场，同Z80一样是公认的功能很强的微处理机芯片。

关于6502内部结构框图，前面已有过叙述。总的说来，6502内部包括两大部分，第一部分是寄存器、运算器。6502内部有六个可供编程的寄存器（A，X，Y，PC，S，P）。运算器是指算术逻辑运算单元ALU，6502的算术、逻辑运算指令诸如加、减、与、

或、异或以及移位、加1、减1等操作皆在ALU中完成。第二部分是控制器，这是个使整个微型机系统按统一时序协调操作的功能部件。它对指令进行读取、译码、执行等操作，它在微处理机、存贮器及I/O接口之间发送控制信号，以便能够实现各部件的协调操作。

为了学习 MPU 6502的指令系统，我们特别要对6502中可供编程的寄存器加以关注。现在将这六个寄存器逐一介绍如下：

1．累加寄存器A

这是个 8 位寄存器，它与算术逻辑运算单元ALU一起完成各种算术逻辑运算，它既可存放操作前的初始数据，又可存放操作结果，所以称为累加器。

2．变址寄存器X

变址寄存器X 是八位寄存器，它在编程中常被当做一个计数器来用。它可以由指令控制而被置成一个常数，并能方便地用加1、减1、比较操作来修改和测试其内容，以使得程序能够方便灵活的处理数据块、表格等问题。

3．变址寄存器Y

它的用法基本同于变址寄存器 X。在有些情况下，例如在程序中要同时处理两个以上的数据块时，一个变址寄存器就显得不够用，所以6502中有两个用于变址的寄存器X和Y。

4．程序计数器PC

它是6502中程序员可访问寄存器中唯一的16位寄存器，PC是用来存放指令地址码的寄存器，由于程序的执行一般为顺序执行方式，每取出一个指令字节后 PC即自动加1，为取下一个指令字节作好准备。所以程序计数器PC中的内容往往是指向下一个指令字节地址。但在执行转移指令时，PC中将被放进要转移的目标地址码。

5．堆栈指针S

它是用来指示堆栈栈顶位置的寄存器，由于6502规定堆栈设置在第1页存贮器中，所以堆栈指针S也是条8位寄存器，只用来指出堆栈位置的低八位地址。S具有在数据进栈操作时自动减1，而在数据出栈操作时自动加1的功能。

6. 标志寄存器P（或称处理机状态寄存器）

这也是条8位寄存器，但是只用其中的7位，第5位空着不用。每条指令在执行之后往往会发生进位溢出，以及结果为全零，或是结果为正数、负数的情况，指令执行后常常需要保留这些情况以做为条件分支的根据，标志寄存器P就是为了适应这种需要而设置的。在P寄存器中设置了以下七个标志位：

7	6	5	4	3	2	1	0 ←位数
N	V	.	B	D	I	Z	C

C——进位标志。指令执行完毕后的最高位进位状态，若最高位有进位则使C置为1，若最高位无进位则使C置为0。

Z——零标志。指令执行完毕后结果为0，则Z被置为1，否则Z被置为0。

I——中断禁止（又称中断屏蔽）标志。此位置0表示准许中断，置1表示禁止中断，但非屏蔽中断请求不受此约束。

D——十进制运算标志。此位置0使6502作二进制运算；此位置1则使6502作十进制运算。（D标志位状态仅仅对于后续的加指令和减指令有作用。）

B——BRK指令标志。此位被置1表示是由于执行了BRK指令而使程序被中止。

V——溢出标志。指令执行完毕后若产生溢出则此标志位被置1（用于有符号数的操作）。

N——负数标志。指令执行完毕后，若结果最高位bit7为1，

表示结果为负数则N标志位被置1；若结果最高位为0，表示结果为正数，N标志被置0。

标志位常常在执行条件转移指令时做为条件判断的依据。标志寄存器中有些标志位可以由置标志位指令进行置位和复位，这将在6502的指令系统中进行介绍。

下面我们对指令执行后如何影响标志位略举二例来加以说明。

例1 两个正数61及4A相加，（61及4A是16进制表示）写出算式来就是

```
  01100001
+ 01001010
──────────
  10101011
```

两个正数相加（两个操作数的符号位都是0）结果为什么成了负数（结果符号位为1）？这是因为 61＋4A＝AB 超过了八位寄存器所能表示的最大正数 7F 而产生了溢出，这就使标志位V置1。结果不是全0所以标志位Z置0。结果最高位为 1，所以标志位N置1。结果最高位没有进位，所以标志位C置0。

例2 －1和＋1两个数相加

用补码运算的算式是

```
  11111111
+ 00000001
──────────
 100000000
```

这个运算结果使得 Z＝1（结果为全0），C＝1（结果最高位有进位），N＝0（结果符号位为0），V＝0(结果无溢出)。

在对6502内部寄存器有了以上的了解之后，我们就可以来介绍6502的寻址方式和指令系统了。

§1—1　6502寻址方式

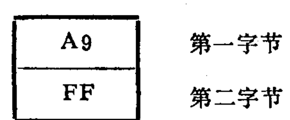
6502的寻址方式共分为十三种（见附录6502指令表）分别介绍如下：

一、立即寻址

采用立即寻址(IMMEDIATE)的指令都是两字节指令。指令的操作数部分给出的不是操作数地址而直接就是操作数本身，我们把它称为立即数。这种寻址方式的指令格式为：

指令格式	字节
操　作　码	第一字节
操　作　数	第二字节

例如取数指令 LDA #$FF，其中#表示后随的是立即数，$是十六进制数表示符号。指令LDA #$FF用十六进制数表示的机器码为：



这条指令的功能是把FF这个立即数送到累加器A。A9和FF是十六进制数。(本文中数的表示，不加说明时皆为十六进制表示)。

二、绝对寻址

采用绝对寻址(ABSOLUTE)的指令皆为三字节指令。指令的操作数部分给出的是操作数在存贮器中的有效地址，称为绝对地址，又称直接地址。它的指令格式为：

指令格式	字节
操　作　码	第一字节
操作数地址低字节	第二字节
操作数地址高字节	第三字节

由于操作数地址是用两个字节(十六位)表示，所以它可以是整个64K 存贮器中的任何一个地址。但是注意，这种寻址方式的指令表示成机器码时，操作数地址是低字节在前，高字节在后。例如取数指令 LDA ＄0300的机器码为：

机器码	
AD	第一字节
00	第二字节
03	第三字节

这条指令的功能是把存贮器0300地址单元中存放的操作数取出来送到累加器A。

三、零页寻址

采用零页寻址（ZERO PAGE）的指令皆为两字节指令。64K存贮器可以按高 8 位地址码划分为256页，高8位为 0 的地址范围称做零页。零页寻址和绝对寻址的区别在于零页寻址方式中操作数的地址仅限于存贮器的零页范围（0000～00FF）所以在表示操作数的地址码时，就无需表示出高 8 位地址。它的指令格式为：

格式	
操　作　码	第一字节
操作数的零页地址	第二字节

例如取数指令 LDA ＄06 的操作数地址只有一个字节06，这就表示它是个零页地址0006，这条指令的机器码为：

机器码	
A5	第一字节
06	第二字节

它的功能是把存贮器0006单元中存放的操作数取出来送到累加器A。

四、累加器寻址

采用累加器寻址(ACCUM.)的指令皆为单字节指令。指令操

作所需要的操作数存在累加器A中，因此这个操作数地址A并不需要单独占用指令字节来表示，而只要隐含在操作码字节之中就行了，所以这类指令就只有一个操作码字节。例如循环左移指令ROL A的机器码为：

2A	一个字节

这条指令的功能是把累加器A中的操作数作一次连同进位C在内的循环左移。

五、隐含寻址

采用隐含寻址(IMPLIED)的指令亦为单字节指令。隐含寻址和累加器寻址两种方式的区别在于隐含寻址中的操作数地址是除去累加器A以外的其它寄存器 X，Y，S或P。这类指令也只有一个操作码字节。例如X减1指令DEX的机器码为：

CA	一个字节

这条指令的功能是把X寄存器中的内容减1后再送回X寄存器（X－1→X）

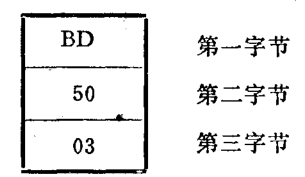
此外还有几条栈操作指令或者和栈操作有关的返回指令也都采用了隐含寻址方式，它们的操作数无需单独占用指令字节。

六、使用X寄存器的绝对变址

为了方便起见，把这种寻址方式简称为绝对X变址——ABS．X。采用绝对X变址方式的指令皆为三字节指令。这种寻址方式是把一个16位绝对地址作为基地址再和一个作为偏移量的X变址寄存器内容相加，经过这样的变址寻址所得的地址才是操作数的有效地址。它的指令格式为：

操　作　码	第一字节
基地址低字节	第二字节
基地址高字节	第三字节

例如取数指令 LDA $0350，X 的机器码为：



这条指令的功能是把0350＋X 单元中的操作数取出来送到累加器A。若X＝06,就把0350＋X＝0356单元中的数取出来送到累加器A。

七、使用Y寄存器的绝对变址

这种寻址方式简称为绝对Y变址——ABS.Y。采用绝对Y变址的指令亦为三字节指令。它和绝对X 变址方式的区别仅在于变址寄存器用的是Y而不是X。它的指令格式为：

指令格式	字节
操　作　码	第一字节
基地址低字节	第二字节
基地址高字节	第三字节

例如取数指令 LDA $6050，Y 的机器码为：

机器码	字节
B9	第一字节
50	第二字节
60	第三字节

这条指令的功能是把6050＋Y 单元中的操作数取出来送到累加器A。

八、使用X寄存器的零页变址

这种寻址方式简称为零页X变址——Z·PAGE，X。采用零页X变址方式的指令皆为两字节指令。它和绝对X变址方式的区别仅在于操作数基地址用的是零页地址而不是绝对地址。它的指令格式为：

操　作　码	第一字节
零页基地址	第二字节

例如取数指令 LDA　$06，X 的机器码为：

B5	第一字节
06	第二字节

这条指令的功能是把0006＋X 单元中的操作数取出来送到累加器A。

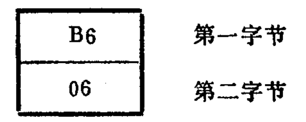
九、使用Y寄存器的零页变址

这种寻址方式简称为零页Y变址——Z·PAGE，Y。采用零页Y变址方式的指令亦为两字节指令。它和绝对Y变址方式的区别仅在于操作数基地址用的是零页地址而不是绝对地址。它的指令格式为：

操　作　码	第一字节
零页基地址	第二字节

要注意的是在6502中采用这种寻址方式的指令只有两条 LDX 和 STX。

例如取数指令 LDX　$06，Y 的机器码为：



B6	第一字节
06	第二字节

这条指令的功能是把0006＋Y 单元中的操作数取出来送到变址寄存器X。

十、间接寻址

间接寻址方式（INDIRECT）在6502中仅仅用于无条件转移JMP这一条指令，是三字节指令。它的操作数部分给出的是操作

数的间接地址，间接地址是指存放操作数有效地址码的地址。它的指令格式为：

操　作　码	第一字节
间接地址低字节	第二字节
间接地址高字节	第三字节

由于操作数有效地址是16位，而每一存贮单元内容只有8位，所以操作数的有效地址必须通过两次间接寻址才能得到。

例如无条件转移指令 JMP（$1000）（指令中的括号即表示间接寻址）。它的机器码为：

6C	第一字节
00	第二字节
10	第三字节

为了求得有效地址，第一次要对1000单元间接寻址得到有效地址

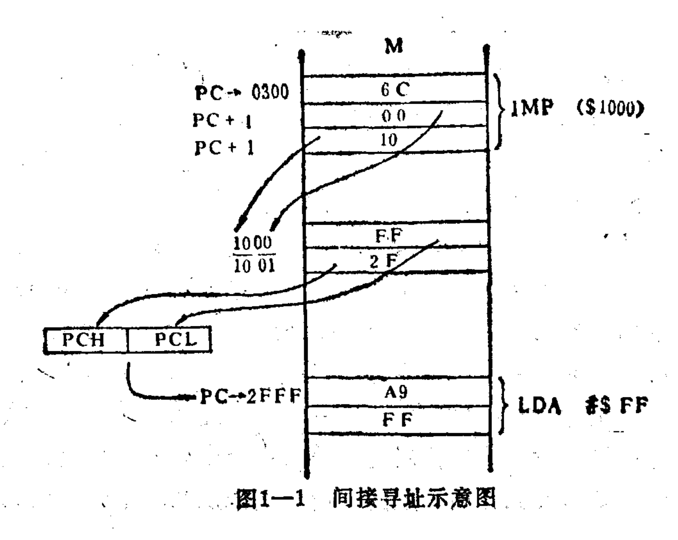


图1—1　间接寻址示意图

低8位，第二次再对1000+1=1001单元间接寻址得到有效地址高8位，把两次间接寻址的结果合在一起就是有效地址。此处若是已有1000单元的内容为＄FF（用(1000)=FF表示），1001单元内容为＄2F(用(1001)=2F表示)，那么有效地址就是2FFF。此条指令JMP(＄1000)的功能就是无条件转移到2FFF地址处。如图1—1所示。放在0300～0302单元中的JMP(＄1000)指令执行完毕后就转到2FFF单元处，接着再执行2FFF～3000单元中放的指令 LDA #＄FF。

十一、相对寻址

相对寻址方式(RELATIVE)只用于条件转移指令，指令长度为两字节。第一字节为操作码，第二字节为条件转移指令的跳转步长，又称为偏移量D。偏移量D的值可正可负，符号位表示在Bit7处，D若为负则用补码表示。它的指令格式为：

指令格式	字节
操 作 码	第一字节
偏 移 量 D	第二字节

相对寻址方式是用本条指令第一个指令字节所在地址和偏移量D相加来得到有效地址。例如指令BCC ＊+5放在存贮器＄0350和＄0351两单元中，那么这条指令的功能是：如果跳转条件成立(C=0)，下条指令的地址（转移地址）就应为0350+5=0355，如果跳转条件不成立，则下条指令地址就应为0350+2=0352。指令中的＊即表示本条指令第一个字节存放的地址)在这里是0350)。由于在取出一条指令后，PC表示的已是下条指令所在地址，所以在取出两字节的条件转移指令后，PC就是已加过2的PC值。例如此例中取出指令 BCC ＊+5后，PC=0302因此在相对寻址方式的符号指令转换成对应的机器码时，第二字节中填的偏移量D应是从加过2的PC值0302开始算起的。此例要求从0350单元跳到0355

单元，故偏移量D应填为03，因为0352＋03＝0355。所以BCC ＊＋5这条指令的机器码为：

机器码	
90	第一字节
03	第二字节

同理，当偏移量D为负时，例如把BCC ＊＋5改为BCC ＊－5仍放在＄0350和＄0351两单元中，那么BCC ＊－5这条指令执行时，若跳转条件成立(C＝0)则转移的目的地址是0350－5＝034B，而计算偏移量D是从加过2的 PC＝0352算起，所以 D＝F9(－7 的补码)，因为0352－7＝034B，可知BCC ＊－7这条指令的机器码为：

机器码	
90	第一字节
F9	第二字节

但是为了避免计算偏移量所带来的麻烦，在使用汇编语言书写源程序时，偏移量都一律用转移目标地址处的符号地址来代替。关于符号地址第二章还将详细叙述。

以上所述寻址过程，如图1—2和图1—3所示。相对寻址方式的跳转步长范围，若从条件转移指令第一个指令字节所在的PC值

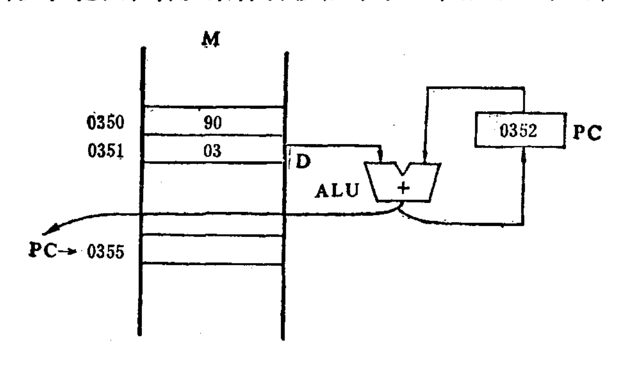


图1—2 偏移量为正的相对寻址示意图

ALU：算术逻辑运算单元

开始计算，则应为：－126～＋129(十进制)。超过这个范围就不能使用相对寻址方式。

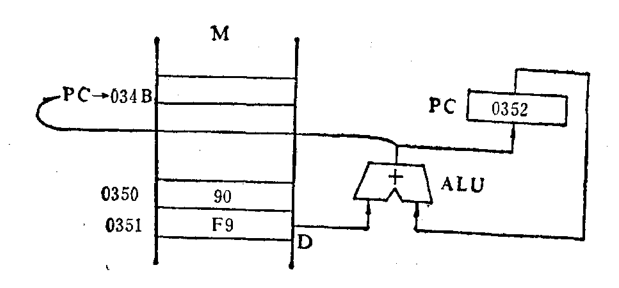


图1—3 偏移量为负的相对寻址示意图

十二、先变址（X）间接寻址

采用先变址（X）间接寻址方式——(IND，X）的指令皆为两字节指令。这种寻址方式是指先以X作为变址寄存器和零页基地址IND相加 IND＋X。但是这个变址计算所得到的还只是一个间接地址，还必须再经过两次间接寻址，才能得到有效地址。第一次是对IND＋X间址得到有效地址低8位，第二次再对IND＋X＋1间址得到有效地址高8位。注意IND只能是个零页地址。这种寻址方式的指令格式为：

操　作　码	第一字节
零页基地址	第二字节

例如取数指令LDA（$06，X）的机器码为：

A1	第一字节
06	第二字节

若已有变址寄存器X＝02，存贮单元（08)＝65，(09)＝87 则根据先变址（X）间接寻址方式规定，执行这条指令的操作数地址低8位是（06＋X)＝(06＋2)＝(08)＝65，操作数地址高8位是

(06 + X + 1) = (06 + 2 + 1) = (09) = 87，由此最后得到了操作数有效地址为：8765。所以这条指令的功能是把8765地址单元中的数取出来送到累加器A。

寻址过程如图1—4所示。注意这种先变址的间接寻址方式中变址寄存器只能用X。

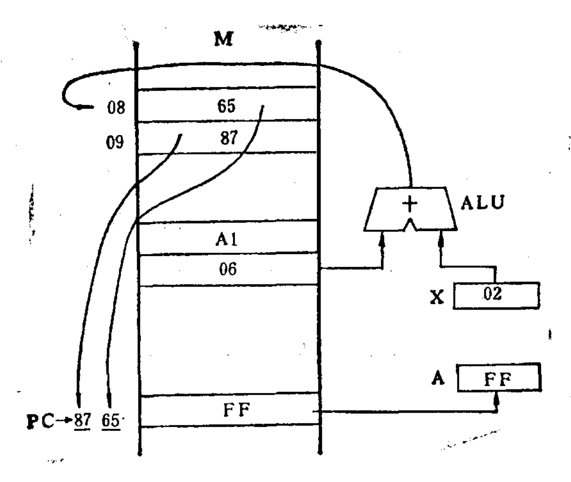
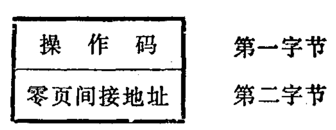


图1—4　先变址（X）间接寻址示意图

由图1—4可见，由于8765单元的内容是FF，所以LDA（$06，X)指令执行之后，A累加器的内容为FF。

十三、后变址（Y）间接寻址

采用后变址（Y）间接寻址方式——(IND)，Y的指令皆为两字节指令。这种寻址方式是指先对IND部分所指出的零页地址作一次间接寻址，得到一个低8位地址，再对IND＋1作一次间接寻址，得到一个高8位地址，最后把这高、低两部分地址合起来做为16位的基地址，并以Y为变址寄存器进行变址计算就得到了操作数的有效地址。注意，IND是个零页地址。这种寻址方式指令的格式为：



例如取数指令LDA （$06），Y的机器码为：

B1	第一字节
06	第二字节

若已有变址寄存器Y＝02，存贮单元(06)＝65，(07)＝87 为了得到有效地址，先要对IND＝06单元作第一次间接寻址，得到(06)＝65，再对IND＋1＝06＋1＝07单元作第二次间接寻址，得到(07)＝87，最后把8765合起来做为基地址，并以Y 为变址寄存器进行变址计算就得到了操作数的有效地址为：8765＋02＝8767。所以这条指令的功能是把8767地址单元中的数取出来送到累加器 A。寻址过程如图1—5所示。注意这种后变址的间接寻址方式中变址寄存器只能是用Y。

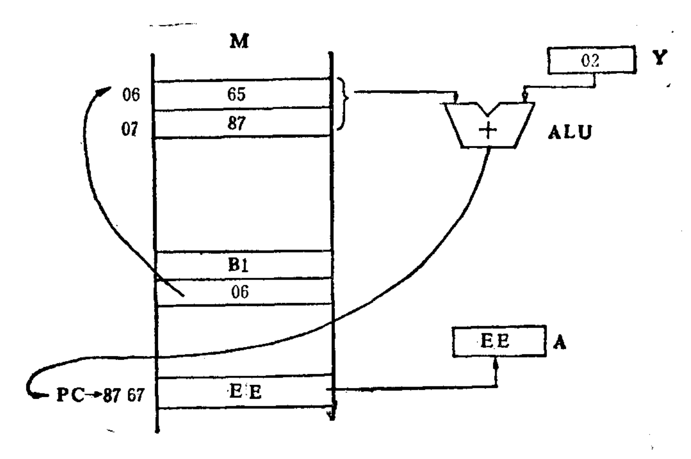


图1—5 后变址（Y）间接寻址示意图

由图1—5可见，由于8767单元内容是EE，所以LDA （$06），

Y指令执行之后，A的内容为EE。

以上十三种寻址方式在编制程序时可供灵活选用。每条指令可以允许使用哪几种寻址方式，请查阅6502指令表。至于在什么情况下选用那种寻址方式，这要根据编制程序的具体需要而定。我们可以做一个概述如下：

立即寻址一般用于设置初始数据。

一般说来，累加器寻址，隐含寻址的操作数多放在6502内部的寄存器中，取操作数不必访问存贮器，所以编制程序时使用这类寻址方式的指令可以加快程序执行速度。

零页寻址由于占用字节少，执行速度较快，因而弥补了6502微处理器内部寄存器数量不多的缺陷，所以在可能情况下，应多使用零页地址。

变址寻址方式用于处理数据块则较为理想。

间接寻址常用来存放专用程序的入口地址，这样就使专用程序块可以方便的在存贮器中浮动，只要将程序块的起始地址放在这种间接寻址方式选用的间接地址中即可。

相对寻址用跳转步长来代替跳转地址，减少了指令的字节长度，缩短了指令执行时间。

后变址(Y) 间接寻址常用于动态数据块处理，即数据块在存贮器中存放的位置是可以变化的，只要把数据块首地址置入这种寻址方式所选用的零页地址中即可。

先变址(X) 间接寻址则用于多个数据块处理。在以下各章所介绍的内容中，可以看到这些寻址方式的具体用法。

应当注意，在遇到变址计算的寻址方式中，有些寻址方式不允许跨页，如零页X 变址，零页Y变址和先变址 (X) 间接寻址。而有些寻址方式则允许跨页，如绝对X变址，绝对Y变址和后变址(Y)间接寻址。此外唯一只被 JMP 指令采用的间接寻址方式也不允许跨页。

§1—2 6502指令系统

6502微处理机共有56条指令，13种寻址方式，它的指令格式很整齐，第一字节皆为操作码，其后就是操作数或操作数地址码。由于每条指令又可对应有不同的寻址方式，因而在实际上一条指令形成了好几种不同的操作码，相当于好几条指令。所以又可认为6502微处理机共有151条指令。见附录：6502指令表。

下面我们把6502的56条指令归纳分类并逐一介绍，对其中的每一条指令不仅要了解其功能而且要了解指令的执行对标志寄存器P所造成的影响。

一、传送指令

1．存贮器和寄存器之间的传送

(1) LDA——由存贮器取数送入累加器 M→A

LDA指令有8种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	0	1	x	x	x	0	1

其中bit2～4这三位就用来表示八种寻址方式。如表1—1所示。至于各种寻址方式对应的指令机器码占有的指令字节数在§1—1中已有过叙述。此处不再重复。

例如：LDA #$F0采用的是立即寻址方式，所以它的机器码第一字节是操作码A9（放在存贮单元nnnn中），第二字节是立即数F0(放在存贮单元 nnnn＋1中)。执行的结果除了将立即数F0送进了累加器A，还对标志寄存器P中的Z和N两个标志位产生影响。此例中被传送的操作数 F0不是一个8位全为0的数，所以Z标志位被置为0状态，又由于F0的最高位是1，认为是一个负数，所以N标志位被置为1。再如LDA #00，由于被传送的操作数是一个8位

表1—1　　　　LDA 对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	A1	先变址(X)间接寻址
0　0　1	A5	零页寻址
0　1　0	A9	立即寻址
0　1　1	AD	绝对寻址
1　0　0	B1	后变址(Y)间接寻址
1　0　1	B5	零页X 变址
1　1　0	B9	绝对Y变址
1　1　1	BD	绝对X变址

全为0的数，所以Z标志位被置为1，而N 标志位被置为0。采用立即寻址的LDA指令功能说明，请看图1—6。

采用绝对寻址，零页寻址，绝对X(或Y)变址寻址，零页 X变址寻址的LDA指令功能说明，依次请看图1—7，图1—8，图1—9，图1—10。采用先变址(X)间接寻址及后变址(Y)间接寻址的LDA指令功能说明在§1—1中已有过叙述，请看图1—4和图1—5。

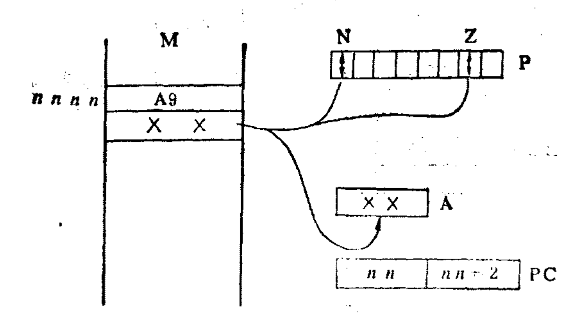


图1—6　采用立即寻址的LDA功能示意图

由图可见，放在nnnn和nnnn＋1单元中的LDA ＃××执行后，立即数××被送进A中。P寄存器中受影响的标志位是N、Z。PC程序计数器内容由nnnn指向下条指令所在地址nnnn＋2。

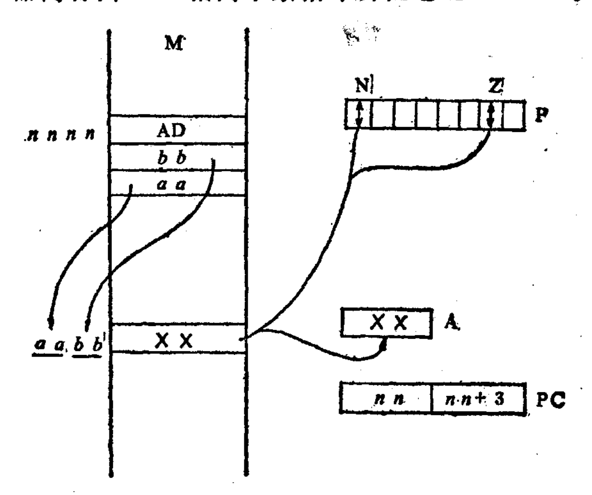


图1—7　采用绝对寻址的LDA功能示意图

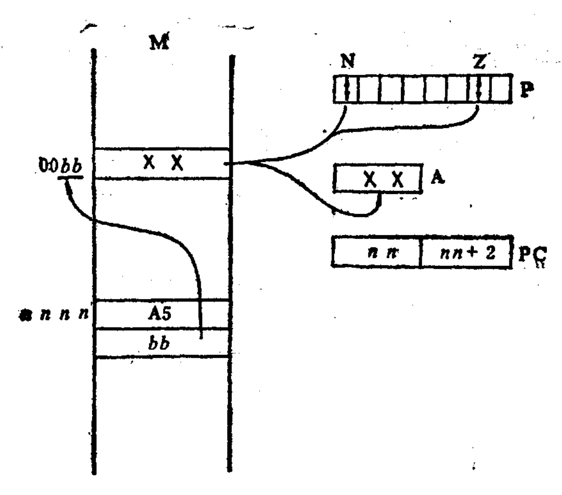


图1—8　采用零页寻址的LDA功能示意图

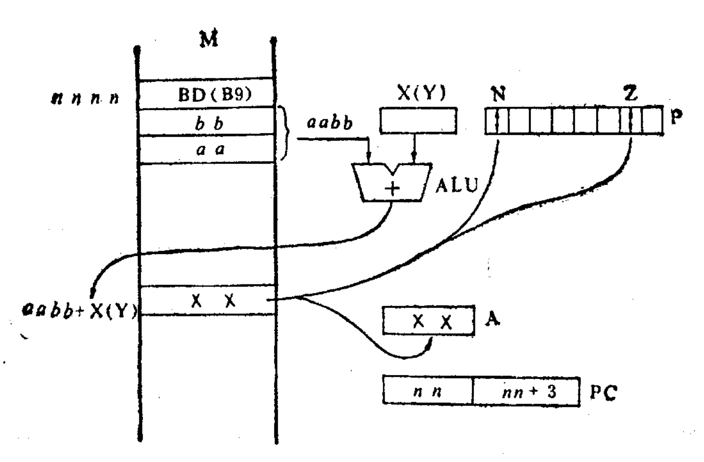


图1—9 采用绝对X(Y)变址的LDA功能示意图

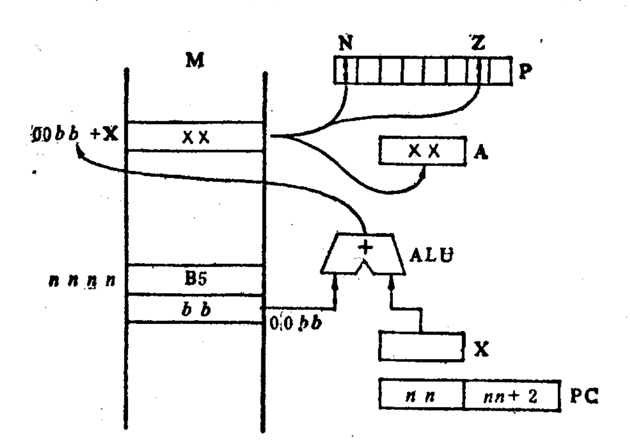


图1—10 采用零页X变址的LDA功能示意图

(2) LDX——由存贮器取数送入变址寄存器X M→X

LDX指令有5种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0	←位数
1	0	1	x	x	x	1	0	

表1—2所示为5种寻址方式对应的操作码。

表1—2　　　　LDX　对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	A2	立即寻址
0　0　1	A6	零页寻址
0　1　1	AE	绝对寻址
1　0　1	B6	零页Y变址
1　1　1	BE	绝对Y变址

下面仅以操作码为BE的绝对Y变址寻址方式为例，说明LDX的指令功能，如图1—11所示。

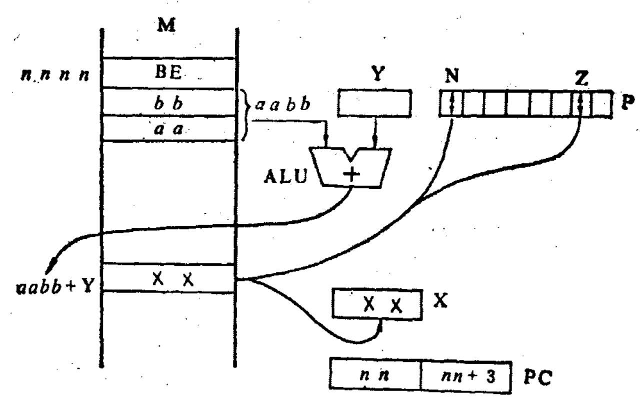


图1—11　采用绝对Y变址的LDX功能示意图

(3) LDY——由存贮器取数送入变址寄存器Y　M→Y

LDY 指令有5种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	0	1	x	x	x	0	0

表1—3所示为5种寻址方式对应的操作码。

表1—3　　　　　　LDY　对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	A0	立即寻址
0　0　1	A4	零页寻址
0　1　1	AC	绝对寻址
1　0　1	B4	零页X变址
1　1　1	BC	绝对X变址

下面仅以操作码为A0的立即寻址方式为例，说明LDY的指令功能，如图1—12所示。

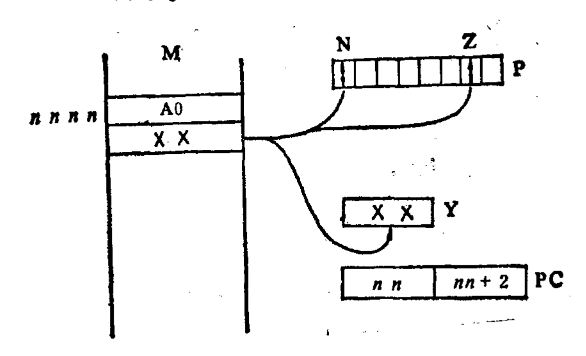


图1—12　采用立即寻址的LDY功能示意图

(4) STA——将累加器的内容送入存贮器　A→M

STA指令有七种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	0	0	x	x	x	0	1

表1—4所示为7种寻址方式对应的操作码。

表1—4　　STA　对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	81	先变址(X)间接寻址
0　0　1	85	零页寻址
0　1　1	8D	绝对寻址
1　0　0	91	后变址(Y)间接寻址
1　0　1	95	零页X变址
1　1　0	99	绝对Y变址
1　1　1	9D	绝对X变址

下面仅以操作码为85的零页寻址方式为例说明STA的指令功能，如图1—13所示。

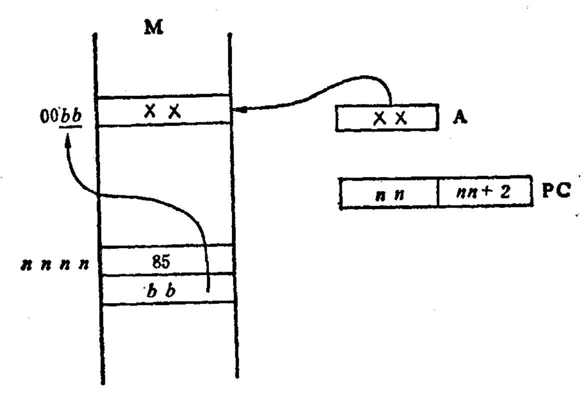


图1—13　采用零页寻址的STA功能示意图

注意：STA指令的执行对标志寄存器没有影响，即标志寄存器P中各个标志位保持执行STA指令之前的原状态不动。

(5) STX——将变址寄存器X的内容送入存贮器　X→M

STX指令有三种寻址方式，它的操作码字节编码如下：

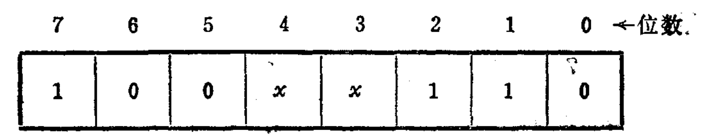


表1—5所示为3种寻址方式对应的操作码。

表1—5　　STX　对应的寻址方式

x　x	指令操作码	寻　址　方　式
0　0	86	零页寻址
0　1	8E	绝对寻址
1　0	96	零页Y变址

下面仅以操作码为96的零页Y变址寻址方式为例，说明 STX 指令功能，如图1—14所示。

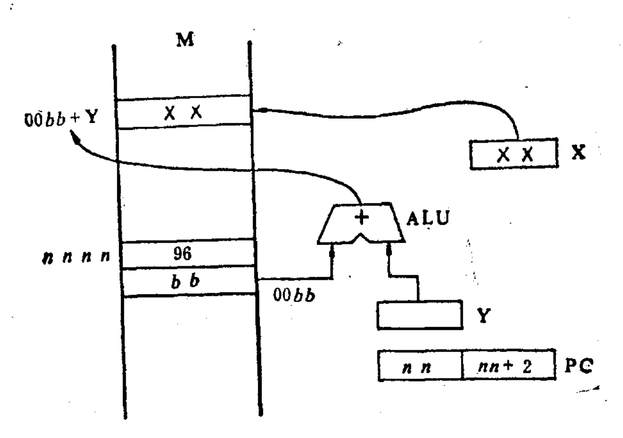


图1—14　采用零页Y变址的STX功能示意图

STX指令亦不影响标志寄存器的状态。

(6) STY——将变址寄存器Y的内容送入存贮器　Y→M

STY 指令也有三种不同的寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	0	0	x	x	1	0	0

表1—6所示为这三种寻址方式对应的操作码。

表1—6　　　　STY　对应的寻址方式

x　　x	指令操作码	寻　址　方　式
0　　0	84	零页寻址
0　　1	8C	绝对寻址
1　　0	94	零页X变址

下面以操作码为8C的绝对寻址方式为例说明 STY 指令的功能，如图1—15所示。

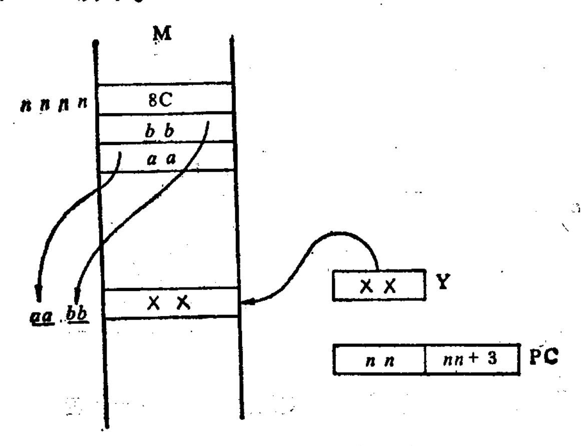


图1—15　采用绝对寻址的STY功能示意图

STY指令不影响标志寄存器P的状态。

2. 寄存器和寄存器之间的传送

这类指令皆为隐含寻址方式的单字节指令，都是在MPU6502内部的寄存器之间交换信息。

(1) TAX——将累加器 A 的内容送入变址寄存器 X A→X 操作码为AA。

(2) TXA——将变址寄存器X的内容送入累加器A X→A 操作码为8A。

(3) TAY——将累加器A的内容送入变址寄存器Y A→Y 操作码为A8。

(4) TYA——将变址寄存器Y的内容送入累加器A Y→A 操作码为98。

(5) TSX——将堆栈指针S的内容送入变址寄存器X S→X 操作码为BA。

以上这五条指令执行后都将影响标志寄存器P中的Z和N标志位。

(6) TXS——将变址寄存器X的内容送入堆栈指针S X→S 操作码为9A。

这条指令执行后不影响标志寄存器P的状态，这是和前五条指令不同的。

3. 传送类指令用法举例

数的传送在计算机中是最基本最大量的操作，因此各种程序的编制都离不开传送类指令。下面仅举一例作为示范。

```
LDA #00
TAY          ; 0→Y
STA $FA      ; 0→00FA单元
STA $FC      ; 0→00FC单元
LDA #$60
```

```
STA  $FB          ; (00FB)(00FA) = 6000
LDA  #$70
STA  $FD          ; (00FD)(00FC) = 7000
  ⋮
LDA  ($FA), Y
STA  ($FC), Y     ; (6000)→7000单元
  ⋮
```

这几条传送指令的执行所完成的功能包括以下四件事：第一件事是使变址寄存器Y的内容为0。第二件事是使 00FB 单元的内容为60，而使 00FA 单元的内容为00，这就把程序的后续部分要用到的一个地址码6000放到了零页地址FB和FA 两单元中，即使(FB)(FA) = 6000。第三件事是使00FD单元内容为70，而使 00FC 单元的内容为00，这就把程序的后续部分要用到的另一个地址码7000放到了零页的FD和FC两单元中，即(FD)(FC) = 7000。第四件事是用LDA ($FA),Y和STA ($FC),Y两条指令实现了将6000单元中的内容传送到7000单元中去。

二、算术逻辑运算指令

1．算术运算指令

(1) ADC——累加器、存贮器、进位标志C相加，结果送累加器 A + M + C→A

ADC指令有8种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	1	1	x	x	x	0	1

表1—7所示为这8种寻址方式对应的操作码。

表1—7　　　　ADC 对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	61	先变址(X)间接寻址
0　0　1	65	零页寻址
0　1　0	69	立即寻址
0　1　1	6D	绝对寻址
1　0　0	71	后变址(Y)间接寻址
1　0　1	75	零页X变址
1　1　0	79	绝对Y变址
1　1　1	7D	绝对X变址

下面以操作码为69的立即寻址方式为例说明 ADC指令功能，如图1—16所示。

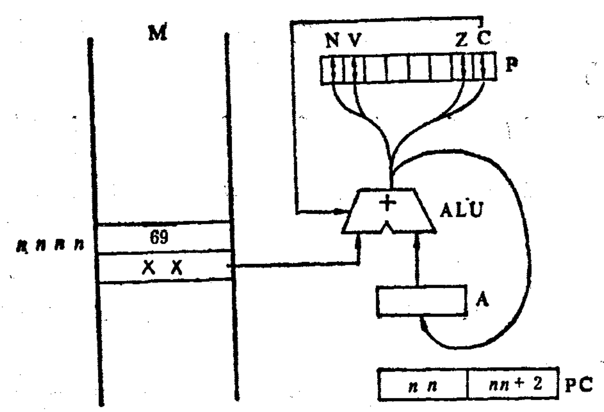


图1—16　采用立即寻址的ADC功能示意图

应当注意，6502的ADC是一条带进位的加法指令。ADC指令的执行将影响标志寄存器P中的四个标志位C、Z、V、N。例如执

行一条ADC ＃＄5A若已有A＝32，C＝1那么ADC ＃＄5A执行的结果应为

$$
\begin{array}{rr}
32= & 00110010 \\
5A= & 01011010 \\
+\ C= & 1 \\
\hline
8D & 10001101
\end{array}
$$

由于相加结果在最高位上无进位，所以使C＝0；由于结果不为0，所以使Z＝0；由于结果最高位为1，所以使N＝1；又由于两个相加数若被认为是有符号数时，则是两个正数相加而结果却超出了一个字节所能表示的最大正数7F而使符号位由0变成了1，产生了溢出，所以使V＝1。因此ADC ＃＄5A 这条指令执行之后使P寄存器中的C＝0，Z＝0，V＝1，N＝1。

2．SBC——从累加器减去存贮器和借位标志$\overline{C}$，结果送累加器 $A-M-\overline{C}\longrightarrow A$

此处所说的借位标志$\overline{C}$其实就是P寄存器中的标志位C取反的状态。

SBC指令也有和ADC指令同样的8种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	1	1	x	x	x	0	1

表1—8所示为这8种寻址方式对应的操作码。

下面以操作码为E5的零页寻址方式为例说明SBC指令功能，如图1—17所示。

SBC 指令的执行亦影响P寄存器中的四个标志C、Z、V、N。应当注意MPU6502在减法操作中用$\overline{C}$表示借位，这是和其它微处理器多用C表示借位不同的。由于6502的SBC是一条带借位的减法指令，所以在只做单字节减法时应预先使借位$\overline{C}$＝0，即预置C＝1

表1—8　　SBC对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	E1	先变址(X)间接寻址
0　0　1	E5	零页寻址
0　1　0	E9	立即寻址
0　1　1	ED	绝对寻址
1　0　0	F1	后变址(Y)间接寻址
1　0　1	F5	零页X变址
1　1　0	F9	绝对Y变址
1　1　1	FD	绝对X变址

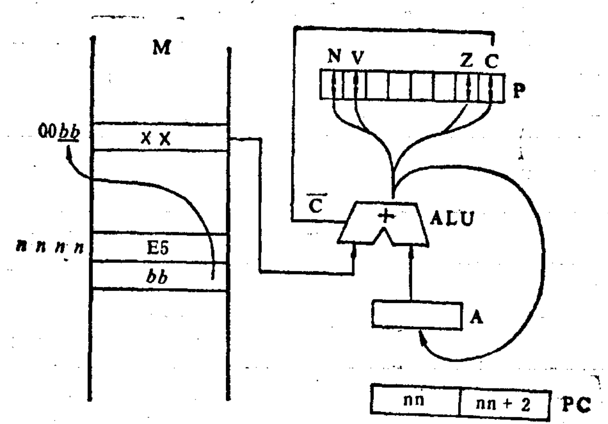


图1—17　采用零页寻址的SBC功能示意图

(这可用下面将介绍的指令SEC来实现)。例如执行一条SBC　$06若已有A=01，(0006)=02，C=1，那么SBC　$06执行的结果应为：

$$
\begin{array}{rr}
01= & 00000001 \\
(02)_{补}= & 11111110 \\
+\bar{C}= & 0 \\
\hline
(-1)_{补} & 11111111
\end{array}
$$

由于相减结果不为0，所以Z=0；由于结果最高位为1，表示结果为负，所以使N=1；由于结果-1未越出一个字节所能表示数的范围，亦即没有溢出，所以使V=0；又由于结果为负，说明不够减，最高位向高字节有借位，所以使C=0(即$\bar{C}$=1有借位)。可知SBC　$06这条指令执行之后使P寄存器中的C=0，Z=0，V=0，N=1。

(3) INC——存贮单元内容增1　M+1→M

INC指令有4种寻址方式，执行结果影响N、Z两标志位。它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	1	1	x	x	1	1	0

表1—9所示为这4种寻址方式对应的操作码：

表1—9　　INC对应的寻址方式

x　　x	指令操作码	寻址方式
0　　0	E6	零页寻址
0　　1	EE	绝对寻址
1　　0	F6	零页X变址
1　　1	FE	绝对X变址

下面以操作码为FE的绝对X变址方式为例，说明INC指令功能，如图1—18所示。

(4) DEC——存贮单元内容减1　M-1→M

DEC指令亦有和INC相同的4种寻址方式，执行结果影响N、

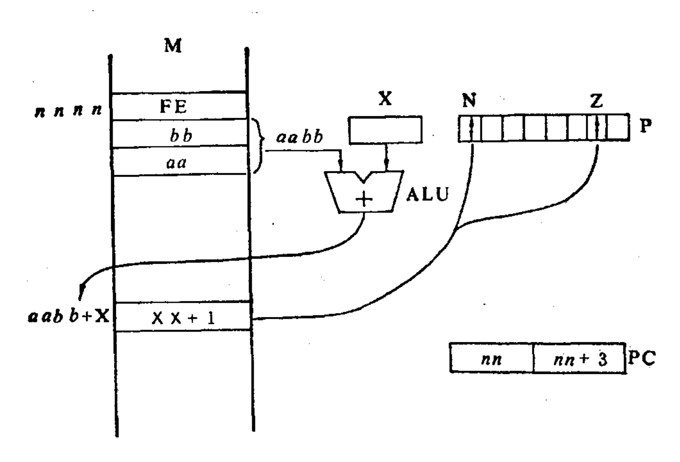
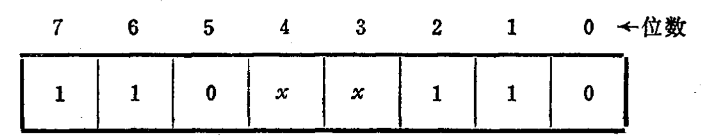


图1—18 采用绝对X变址的INC功能示意图

Z两标志位。它的操作码字节编码如下：



7	6	5	4	3	2	1	0 ←位数
1	1	0	x	x	1	1	0

表1—10所示为这4种寻址方式对应的操作码：

表1—10　　　　DEC 对应的寻址方式

x　　x	指令操作码	寻址方式
0　　0	C6	零页寻址
0　　1	CE	绝对寻址
1　　0	D6	零页X变址
1　　1	DE	绝对X变址

下面也以绝对X变址方式(操作码为DE)为例，说明DEC指令功能，如图1—19所示。

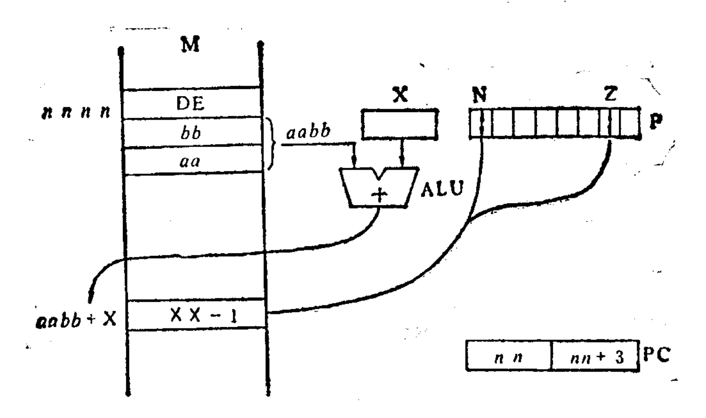


图1—19 采用绝对X变址的DEC功能示意图

(5) INX——变址寄存器X内容增1 X+1→X

是采用隐含寻址方式的单字节指令，操作码为E8，影响P寄存器的N、Z两标志。

(6) DEX——变址寄存器X内容减1 X-1→X

是采用隐含寻址方式的单字节指令，操作码为CA，影响P寄存器的N、Z两标志。

(7) INY——变址寄存器Y内容增1 Y+1→Y

是采用隐含寻址方式的单字节指令，操作码为C8，影响N、Z两标志。

(8) DEY——变址寄存器Y内容减1 Y-1→Y

是采用隐含寻址方式的单字节指令，操作码为88，影响N、Z两标志。

2. 逻辑运算指令

(1) AND——存贮器同累加器相与，结果送累加器 A∧M→A

AND指令有8种寻址方式，它的操作码字节编码如下：

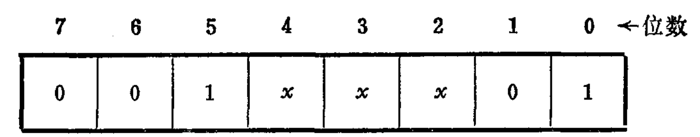


表1—11　　AND　对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	21	先变址(X)间接寻址
0　0　1	25	零页寻址
0　1　0	29	立即寻址
0　1　1	2D	绝对寻址
1　0　0	31	后变址（Y）间接寻址
1　0　1	35	零页X变址
1　1　0	39	绝对Y变址
1　1　1	3D	绝对X变址

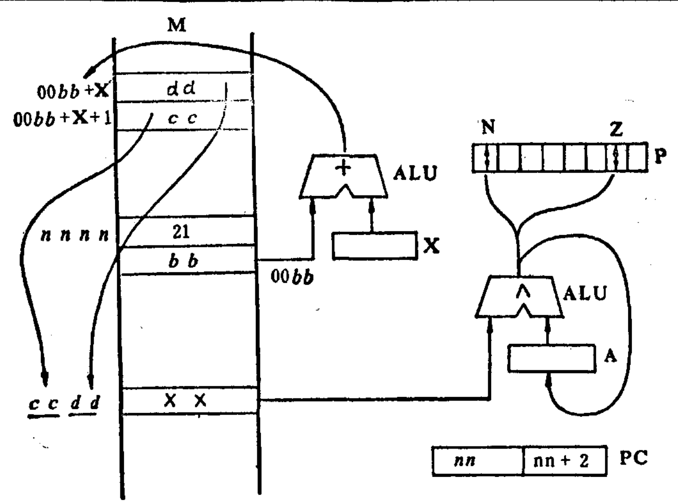


图1—20　采用先变址(X)间接寻址的AND功能示意图

表1—11所示为这8种寻址方式对应的操作码。

下面以操作码为21的先变址（X）间接寻址方 式为例，说明AND指令功能，如图1—20所示。

(2) ORA——存贮器同累加器相**或**结果送累加器　A∨M→A

ORA指令亦有8种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	0	0	x	x	x	0	1

表1—12所示为这8种寻址方式对应的操作码。

表1—12　　ORA　对应的寻址方式

$x\ x\ x$	指令操作码	寻址方式
0 0 0	01	先变址(X)间接寻址
0 0 1	05	零页寻址
0 1 0	09	立即寻址
0 1 1	0D	绝对址
1 0 0	11	后变址(Y)间接寻址
1 0 1	15	零页X变址
1 1 0	19	绝对Y变址
1 1 1	1D	绝对X变址

下面以操作码为19的绝对Y变址方式为例，说明ORA指令功能，如图1—21所示。

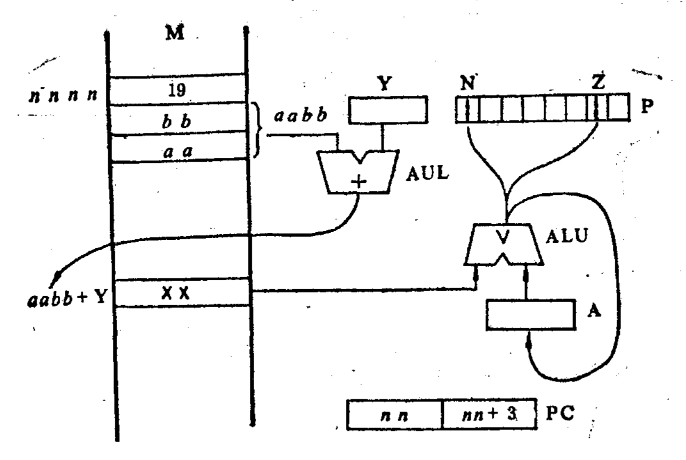


图1—21　采用绝对Y寻址的ORA功能示意图

（3）EOR——存贮器同累加器异或，结果送累加器A∀M→A

EOR指令对应的8种寻址方式同AND，ORA对应的8种寻址方式是相同的，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	1	0	x	x	x	0	1

表1—13所示为这8种寻址方式对应的操作码。

下面以操作码为51的后变址（Y）间接寻址方式为例，说明EOR指令功能，如图1—22所示。

这三条逻辑运算指令都影响标志位N和Z。注意，逻辑运算操作各位之间是独立进行的，彼此之间没有关系，不象算术运算指令相邻位之间有进位、借位关系。

3．算术逻辑运算类指令用法举例

关于MPU6502加法指令和减法指令的用法应当注意到它们是带进位的二进制加法和带借位的二进制减法。用以下例子来说

表1—13　　EOR 对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	41	先变址(X)间接寻址
0　0　1	45	零页寻址
0　1　0	49	立即寻址
0　1　1	4D	绝对寻址
1　0　0	51	后变址(Y)间接寻址
1　0　1	55	零页X变址
1　1　0	59	绝对Y变址
1　1　1	5D	绝对X变址

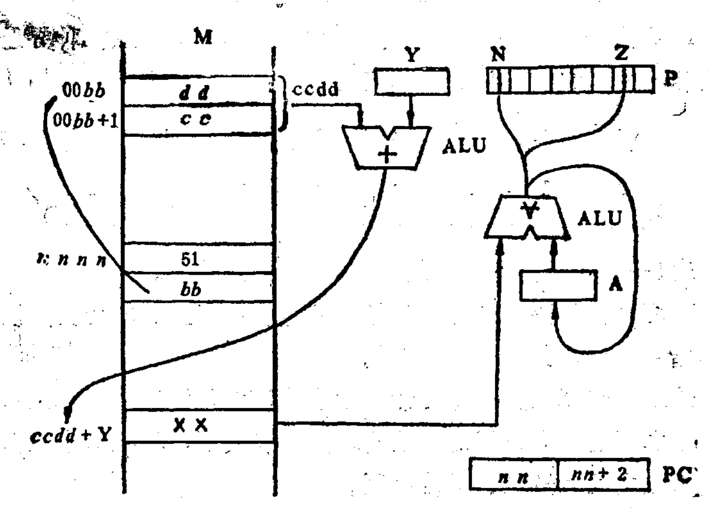


图1—22　采用后变址(Y)间接寻址的EOR功能示意图

明它们如何用于编程。

(1) 八位二进制加法

```
CLD
CLC
LDA  $6000
ADC  $6001
STA  $6002
 ⋮
```

这几条指令构成的一段小程序实现了把6000单元中的八位二进制数和6001单元中的八位二进制数相加，并将结果送入6002单元。例如，(6000)=38，(6001)=2B，则程序执行后(6002)=63。由于ADC是带进位加，因此作两个八位数相加时，应将进位标志C清0，这就是在用ADC指令之前多了一条CLC指令的原因，CLC指令的功能是将标志C清0。那么为什么还用了一条CLD指令呢？前面已经介绍过，D是标志寄存器P中的十进制运算标志位，D置1表示后续的加减指令是作十进制加减运算，D置0表示后续的加减指令是作二进制加减运算。此例是要求作二进制加法，因此除了要在ADC之前使用一条CLC之外，还要使用一条CLD，以通过CLD指令的执行将标志位D置0。

(2) 八位二进制减法

```
CLD
SEC
LDA  $6000
SBC  $6001
STA  $6002
 ⋮
```

这段小程序实现了从6000单元中的八位二进制数减去6001单元中的八位二进制数，并将结果存入6002单元。例如(6000)=77，(6001)=39，则程序执行后(6002)=3E。由于SBC是带借位减，6502规定$\overline{C}$表示借位，当$\overline{C}=0$（即C=1)表示无借位。因此在使用

SBC 指令之前要用一条 SEC 指令将标志C置1，至于又用了一条 CLD 指令的原因在于此例要作的是二进制减法而不是十进制减法。

(3) 两位十进制数加法

```
CLC
SED
LDA  $6000
ADC  $6001
STA  $6002
 ⋮
```

这段小程序实现了把6000单元中的两位十进制数（BCD码）和6001单元中的两位十进制数相加，结果送入6002单元保存。例如（6000）= 26，（6001）= 15，则程序执行后有（6002）= 41。由于本例要作的是两位用BCD码表示的十进制数相加，因此除了在ADC指令之前要用CLC之外，还要使用SED指令预先将十进制运算标志D置1，这样才能进行十进制加法（而不是二进制加法）而使得到的结果为十进制数41。

(4) 两位十进制数减法

```
SEC
SED
LDA  $6000
SBC  $6001
STA  $6002
 ⋮
```

这段小程序实现了由6000单元中的两位十进制数减去6001单元中的两位十进制数，结果存入6002 单元。例如（6000）= 31，（6001）= 15，则程序执行后有(6002) = 16。由于是两个单字节中的数相减，所以在SBC之前要使用SEC以保证借位为0，又由于是

十进制数减法，所以在SBC之前还要使用SED，以保证相减结果是十进制数。

由以上四例可见，加减法编程应注意以下两点：

①由于6502的加法指令带进位，减法指令带借位，加法运算C＝0表示无进位，减法运算 C＝1 表示无借位，所以在执行单字节加法运算之前应使用SEC以清除进位，而在进行单字节减法运算之前应使用SEC以清除借位。

②在作二进制加法或减法运算之前，应使用CLD将十进制运算标志位D 置成 0 状态，而在作十进制加法或减法之前，应使用SED将十进制运算标志D置成1状态。

（5） 十六位二进制加法

```
CLD
CLC
LDA  $6000
ADC  $6002
STA  $6004
LDA  $6001
ADC  $6003
STA  $6005
  ⋮
```

这段程序实现了将6001及6000两单元中放的共计十六位二进制数同6003及6002两单元中放的十六位二进制数相加，结果存入6005及6004两单元中。其中高八位是放在奇数地址码中，例如(6001)＝67，(6000)＝2A，(6003)＝14，(6002)＝F8，则程序执行后有(6005)＝7C，(6004)＝22

即
```
   672A
 + 14F8
 ------
   7C22
```

由程序我们可以看到，在执行6000单元和6002单元中的两个低八位数相加之前是用CLC将进位C清0的，而在进行6001单元和6003单元中的两个高八位数相加之前却不能再使用CLC指令，进位C的状态将由低八位相加结果而定(例如此例中，低八位相加后C=1)，并将参加高八位的相加运算。所以带进位的加法指令ADC及带借位的减法指令SBC用于多字节运算是很方便的。

(6) 屏蔽高四位

```
LDA  $6000
AND #$0F
STA  $6001
 ⋮
```

这三条指令实现了将6000单元内容的低四位送入6001单元，而将6001单元中的高四位清除(屏蔽)。例如(6000)=3D，则这三条指令执行之后有(6001)=0D。

(7) 求反码

```
LDA  $6000
EOR #$FF
STA  $6001
 ⋮
```

MPU 6502中没有求反码的指令，但是这三条指令就完成了将6000单元中的数求反码的功能，并将结果送入6001单元保存。例如(6000)=6A，执行这三条指令后就有(6001)=95。95(10010101)就是6A(01101010)的反码。

三、置标志位指令

这类指令皆为隐含寻址的单字节指令。

1. CLC——清除进位标志C　0→C

操作码为18，影响P寄存器中的C标志位。

2. SEC——置位进位标志C　1→C

操作码为38，影响P寄存器中的C标志位。

3．CLD——清除十进制运算标志D　0→D

操作码为D8，影响P寄存器中的D标志位。

4．SED——置位十进制运算标志D　1→D

操作码为F8，影响P寄存器中的D标志位。

5．CLV——清除溢出标志V　0→V

操作码为B8，影响P寄存器中的V标志位。

注意6502中仅有清除V标志指令，而无置位V标志指令。

6．CLI——清除中断禁止标志I　0→I

操作码为58，影响P寄存器中的I标志位。当外部设备请求中断信号送到6502引腿$\overline{IRQ}$时，只有P寄存器中 I 标志位为0状态，6502才会响应外部设备的中断请求而转入中断服务程序。应当注意，I是中断禁止标志而不是中断允许标志，只有I＝0，6502微处理机才处于开中断状态，亦即中断未被禁止的状态。

7．SEI——置位中断禁止标志I　1→I

操作码为78，影响P寄存器中的I标志位。当标志位I＝1时，6502微处理机处于关中断状态，亦即中断被禁止的状态，则外部设备送到6502的中断请求信号将不被响应。

四、比较指令

1．CMP——累加器同存贮器比较　A－M

CMP指令也是作减法操作 A－M，但是它同减法指令有两点区别，一是借位标志$\overline{C}$不参加减法运算，因此使用CMP 指令之前不必置位 C 标志。二是减法运算的结果不送入累加器A，即指令CMP执行之后不改变A的内容，仅影响P寄存器中的标志位C、Z、N，尤应注意它如何影响标志位C。若执行 CMP之后，C＝1则表示够减无借位，即 A≥M。若执行 CMP 之后，C＝0 则表示不够减，有借位，即A<M，从而由C的状态判断出A同M两数谁大谁小。

CMP指令有8种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	1	0	x	x	x	0	1

表1—14所示为这8种寻址方式对应的操作码。

下面以操作码为C5的零页寻址方式为例，说明 CMP 指令功能。如图1—23所示。

表1—14　　CMP 对应的寻址方式

x　x　x	指令操作码	寻　址　方　式
0　0　0	C1	先变址(X)间接寻址
0　0　1	C5	零页寻址
0　1　0	C9	立即寻址
0　1　1	CD	绝对寻址
1　0　0	D1	后变址(Y)间接寻址
1　0　1	D5	零页X变址
1　1　0	D9	绝对Y变址
1　1　1	DD	绝对X变址

2. CPX——X变址寄存器同存贮器比较　X－M

这条指令和CMP指令的功能相似，只不过把累加器A换成X。另外CPX对应的寻址方式只有3种。它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	1	1	0	x	x	0	0

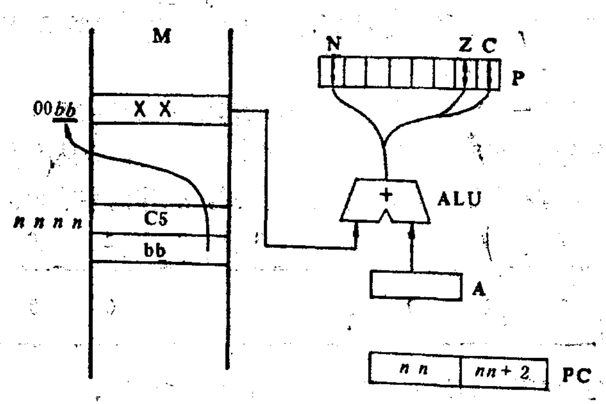


图1—23 采用零页寻址的CMP功能示意图

表1—15所示为这3种寻址方式对应的操作码。

表1—15 CPX 对应的寻址方式

x x	指令操作码	寻 址 方 式
0 0	E 0	立即寻址
0 1	E 4	零页寻址
1 1	EC	绝对寻址

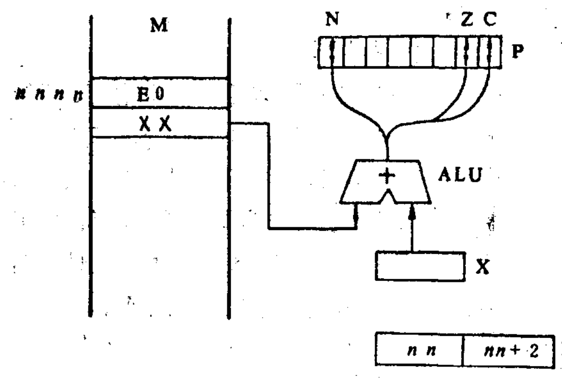


图1—24 采用立即寻址的CPX功能示意图

下面以操作码为E0 的立即寻址方式为例，说明 CPX 指令功能，如图1—24所示。

3．CPY——Y变址寄存器同存贮器比较　Y－M

这条指令和CPX指令的功能相同，只不过把变址寄存器X 换为Y。它也有和CPX相同的3种寻址方式，操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
1	1	0	0	x	x	0	0

表1—16所示为这3种寻址方式对应的操作码。

表1—16　　　CPY　对应的寻址方式

x　　x	指令操作码	寻址方式
0　　0	C0	立即寻址
0　　1	C4	零页寻址
1　　1	CC	绝对寻址

下面以操作码为CC的绝对寻址方式为例，说明 CPY 指令功能，如图1—25所示。

4．BIT——位测试　A∧M

这条位测试指令的功能和AND指令有相同之处，那就是把累加器A同存贮单元内容相与，但是与AND 指令不同的是相与的结果并不送入累加器，亦即 BIT 指令执行结果不改变累加器及存贮器的内容。此外BIT指令对标志寄存器P的影响也和 AND 指令不同。BIT指令对P寄存器Z，N，V三个标志位的影响如下：

Z＝1　如果A∧M＝0；Z＝0　如果 A∧M≠0

N＝M的第7位

V＝M的第6位

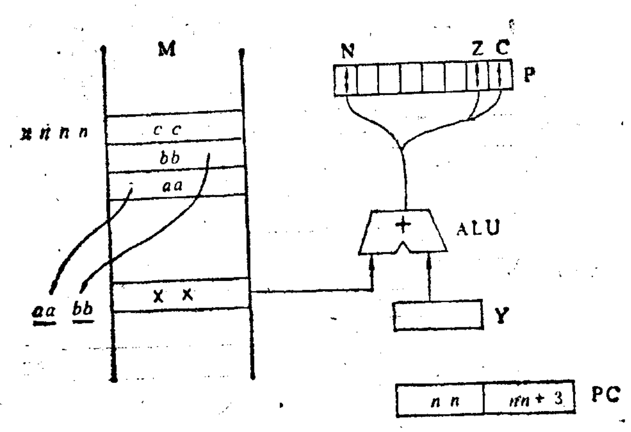


图1—25 采用绝对寻址的CPY功能示意图

所以执行BIT指令后N、V两标志位的状态就是参加与操作的存贮单元内容的最高两位状态。BIT 指令只有零页寻址和绝对寻址两种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	0	1	0	x	1	0	0

表1—17所示为这两种寻址方式对应的操作码。

表1—17　　BIT 对应的寻址方式

x	指令操作码	寻址方式
0	24	零页寻址
1	2C	绝对寻址

下面以操作码为24的零页寻址方式为例，说明BIT指令功能，如图1—26所示。

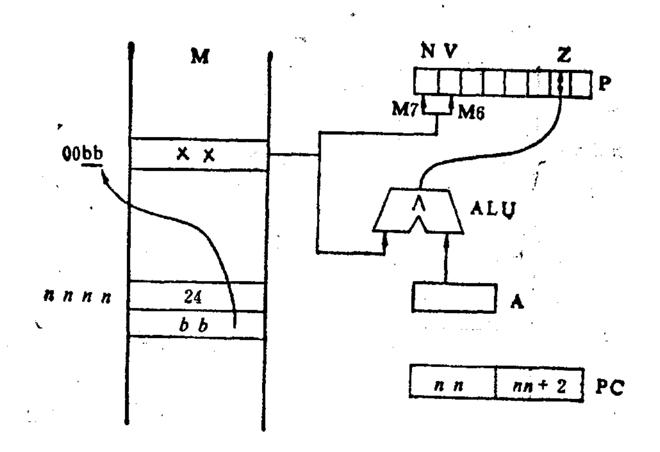


图1—26　采用零页寻址的BIT功能示意图

比较类指令常和条件转移类指令结合在一起用于编程，以下介绍条件转移指令之后再举例说明它们的用法。

五、移位指令

1. 算术左移指令ASL

ASL的移位功能是将字节内各位依次向左移一位，最高位移进标志位C中，最低位补0。称为算术左移的原因在于此条指令执行结果相当于把移位前的数乘2。

ASL指令有5种寻址方式，它的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	0	0	x	x	x	1	0

表1—18所示为这5种寻址方式对应的操作码。

图1—27所示为累加器寻址方式的ASL指令功能。

由于最高位移进了C，所以ASL指令当然要影响P寄存器中的C标志位，另外它还影响N、Z标志。

表1—18　　ASL 对应的寻址方式

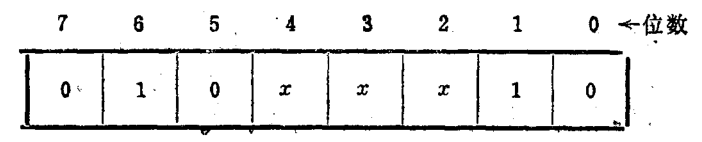
x　x　x	指令操作码	寻址方式
0　1　0	0A	累加器寻址
0　0　1	06	零页寻址
0　1　1	0E	绝对寻址
1　0　1	16	零页X变址
1　1　1	1E	绝对X变址



图1—27　ASL　A　指令功能示意图

2．逻辑右移指令 LSR

LSR的移位功能是将字节内各位依次向右移一位，最低位移进标志位C中，最高位补0。这个操作对于无符号数或者正数相当于乘$\frac{1}{2}$，但对于负数却不能相当于乘$\frac{1}{2}$的操作，所以不能称为算术右移而称为逻辑右移。LSR亦有和ASL相同的5种寻址方式，它的操作码字节编码如下：



7	6	5	4	3	2	1	0 ←位数
0	1	0	x	x	x	1	0

表1—19所示为这5种寻址方式对应的操作码。

图1—28所示为累加器寻址方式的LSR指令功能。

由于最低位移进了C，所以LSR指令当然要影响P寄存器中的C标志位，由于最高位移进了0，所以使标志位N＝0，此外它还影响Z标志位。

表1—19　　LSR　对应的寻址方式

x　x　x	指令操作码	寻址方式
0　1　0	4A	累加器寻址
0　0　1	46	零页寻址
0　1　1	4E	绝对寻址
1　0　1	56	零页X变址
1　1　1	5E	绝对X变址



图1—28　LSR　A　指令功能示意图

3．循环左移指令ROL

ROL的移位功能是将字节内容各位连同进位标志C一起依次向左移一位，它亦有5种寻址方式，操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	0	1	x	x	x	1	0

表1—20所示为这5种寻址方式对应的操作码。

图1—29所示为累加器寻址方式的ROL指令功能。

ROL指令的执行影响标志位N、Z、C。

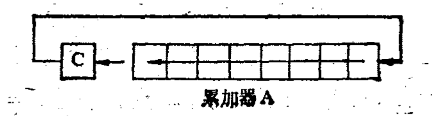


图1—29　ROL　A　指令功能示意图

表1—20　　ROL　对应的寻址方式

x　x　x	指令操作码	寻址方式
0　1　0	2A	累加器寻址
0　0　1	26	零页寻址
0　1　1	2E	绝对寻址
1　0　1	36	零页X变址
1　1　1	3E	绝对X变址

4. 循环右移指令ROR

ROR的移位功能是将字节内各位连同标志位C一起依次向右移一位，它亦有5种寻址方式，操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	1	1	x	x	x	1	0

表1—21所示为这5种寻址方式对应的操作码。

表1—21　　ROR　对应的寻址方式

x　x　x	指令操作码	寻址方式
0　1　0	6A	累加器寻址
0　0　1	66	零页寻址
0　1　1	6E	绝对寻址
1　0　1	76	零页X变址
1　1　1	7E	绝对X变址

图1—30所示为累加器寻址方式的ROR指令功能。

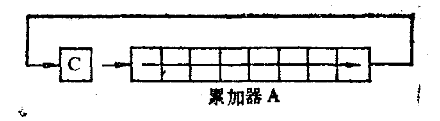


图1—30　ROR　A　指令功能示意图

ROR指令的执行影响标志位N、Z、C。

5. 移位指令用法举例

(1) 拆字

欲将存贮单元6000的内容拆成两段，每段四位，并将6000单元的高四位放入6001单元的低四位，将6000单元的低四位放入6002单元的低四位，清除6001和6002单元的高四位。

```
存贮器地址   指令
0300        LDA  $6000   ；取出操作数
0303        AND  #$0F    ；屏蔽高四位
0305        STA  $6002   ；存结果
0308        LDA  $6000   ；再取操作数
030B        LSR  A       ；将高四位挪到低
                           四位，高四位补0
030C        LSR  A
030D        LSR  A
030E        LSR  A
030F        STA  $6001   ；存结果
0312        BRK
```

此程序中连续使用了四条LSR指令将操作数的高四位挪到了低四位的位置上，并且使高四位都为0。

(2) 将十六位无符号二进制数乘以$\frac{1}{2}$。例如将6000单元和6001单元中的数连接起来作一次算术右移。

这只要用以下两条指令就可以实现。

```
LSR    $6000
ROR    $6001
```

这两条指令的联合移位情况可用图1—31表示。

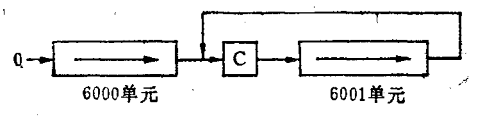


图1—31 十六位二进制数算术右移示意图

两条指令的执行除了把6000和6001两个单元中的数都逐位向右移了一位外，还把移进C中的6000单元最低位移进了6001单元的最高位，而且6000单元的最高位被移进了0，这样就等于完成了一次16位无符号二进制数乘$\frac{1}{2}$的操作。

六、堆栈操作指令

在介绍堆栈操作指令之前，我们首先简单介绍一下堆栈。

微型机通常要在随机存贮器RAM中开辟某个区域用于一些重要数据的暂存。但是对于这个存贮区中的数据如何进行存取却和存贮器中其它区域有着不同的规律。它必须遵从“后进先出”(或称“先进后出”)的规则，那么这块存贮区域就叫做堆栈。例如，因某种需要已经逐个依序存进堆栈中的数据(称为进栈)，在把这些数据再从堆栈中又依序逐个取出时（称为出栈)。必须是先取最后进栈的那个数据，而最先进栈的那个数据却应最后一个出栈，这就叫做“后进先出”。做个通俗的比喻，这就好象餐馆中洗碟子一样，洗的时候是把洗净的碟子从下向上逐个堆放起来，等到用碟子的时候则是从上向下逐个取用。完全符合“后进先出”的规则。

和有些微处理机不同，MPU6502规定堆栈设置在第1页位置上（第1页是指地址码高字节为01的存贮区域)，即只能把0100～01FF这256个单元用作堆栈。为了对堆栈中的数据进行存取操作，

还必须有一个作用类似程序计数器 PC 的堆栈指针 S，它是MPU 6502中的一个八位计数器，其作用是指示堆栈中允许进行存取操作的当前地址。由于6502规定堆栈设置在1页，所以S只需八位就

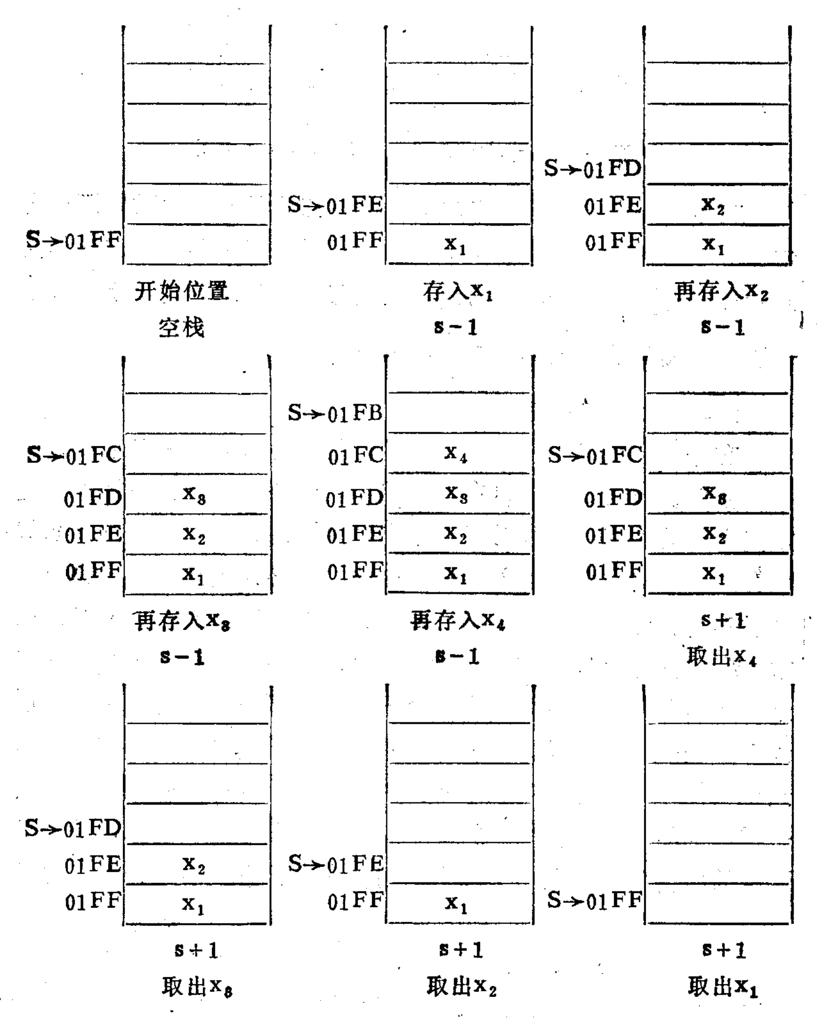


图1—32　堆栈操作过程示意(一)

够了，即S表示的是堆栈地址的低八位，而高八位固定为01。又由于堆栈操作的规则是"后进先出"，所以堆栈指针S在出栈操作时必须具有自动加1功能，而在进栈操作时必须具有自动减1的功能。

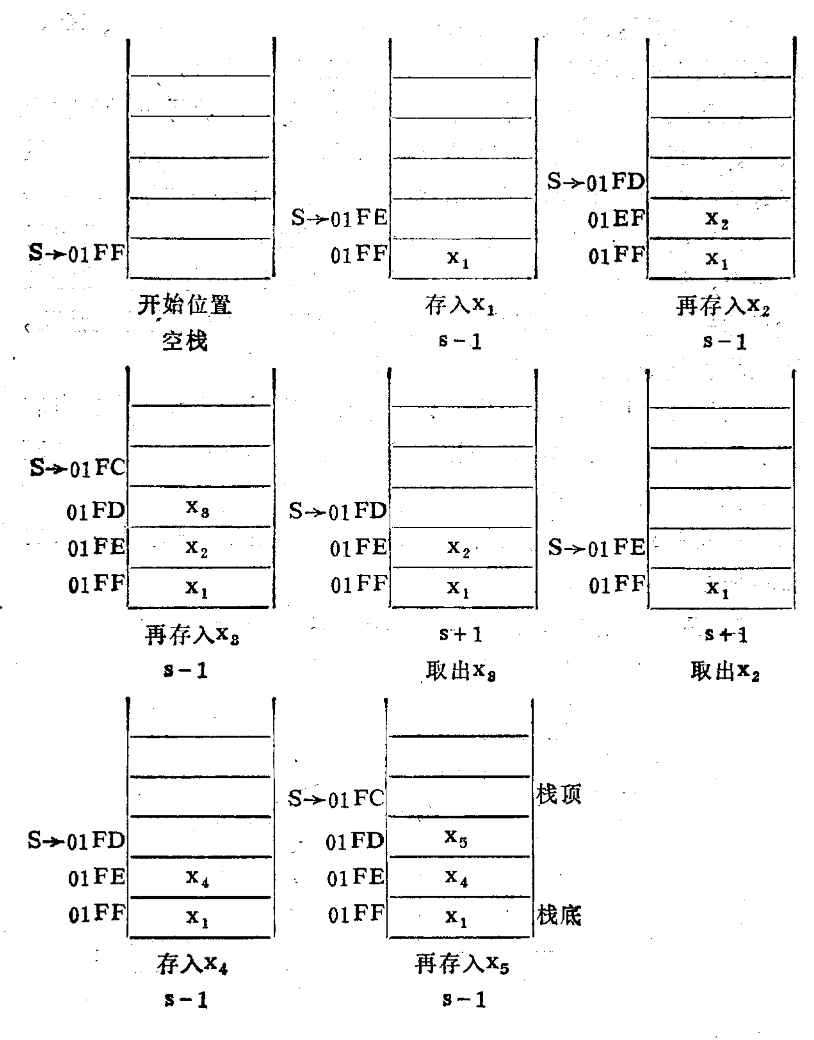


图1—33 堆栈操作过程示意(二)

我们用下面两个例子来具体说明数据进栈和出栈的过程。

例1 将X_1、X_2、X_3、X_4四个数据顺序存入堆栈，然后再按逆序从堆栈中将这四个数据取出，操作过程如图1—32所示。

例2 将X_1、X_2、X_3三个数依次存入堆栈后取出X_3、X_2，然后再顺序存入X_4和X_5。操作过程如图1—33所示。

位于栈中允许进出的那个单元称为栈顶，栈的另一端则称为栈底。栈底占据堆栈中最高地址码单元，而栈顶则占据堆栈中最低地址码单元。由堆栈操作示意图图1—32和图1—33可知，数据的进栈和出栈总是用栈指针S指明堆栈的具体地址，而且在进出栈操作中，栈指针要自动进行减1和加1操作，以使栈指针S始终指向栈顶的一个空单元。这样在堆栈指令中就无须给出堆栈的地址码，而使堆栈指令很短，可以自动形成地址。

所以堆栈用于转子程序，以及中断处理过程中的保存断点和恢复断点是很方便的，在子程序及中断服务程序中还需要保护现场和恢复现场，这些都要使用堆栈指令来实现。这部分内容将在以后的有关章节中进行较为详细的介绍。现在我们来介绍6502的四条堆栈操作指令。

1．累加器进栈指令PHA——A→Ms　S－1→S

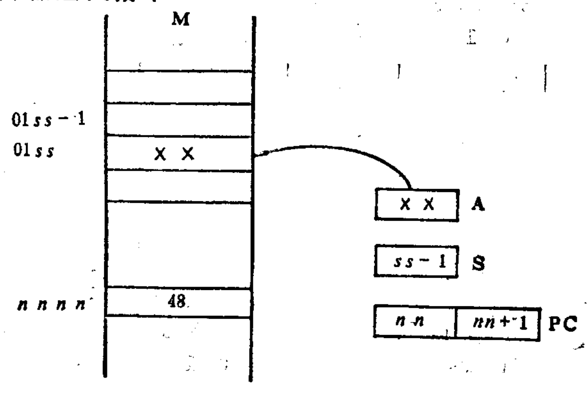


图1—34　PHA指令功能示意图

PHA是隐含寻址方式的单字节指令，操作码为48。它的功能是把累加器A的内容按栈指针S所指示的位置01ss送入堆栈，然后再把栈指针减1 (S－1→S)，这条指令不影响标志寄存器P的状态。图1—34为其功能示意图。

2．标志寄存器进栈指令PHP——$P \to M_S$ $S-1 \to S$

PHP是隐含寻址方式的单字节指令，操作码为08。它的功能是把标志寄存器P的内容按栈指针S所指的位置（01ss）送入堆栈，然后再把栈指针减1 (S－1→S)。这条指令对标志寄存器P本身的状态无影响。图1—35为PHP指令功能示意图。

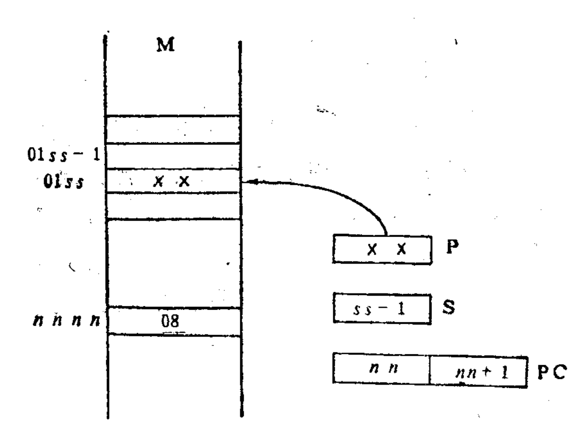


图1—35　PHP 指令功能示意图

3．累加器出栈指令PLA——　$S+1 \to S$　$M_S \to A$

PLA为隐含寻址方式的单字节指令，操作码为68。由于MPU 6502栈指针S是指向栈顶处一个空单元，所以 PLA 的操作功能是首先使栈指针加1 (S＋1→S)，然后再取加过1的 S 所指向单元中的内容并把它送入累加器A 中。这里应注意进栈指令的操作是先进栈然后把栈指针减1。而出栈指令的操作是先将栈指针加1，然后再出栈。PLA指令的执行将影响标志寄存器P中的N、Z两标志

位状态。图1—36为PLA指令功能示意图。

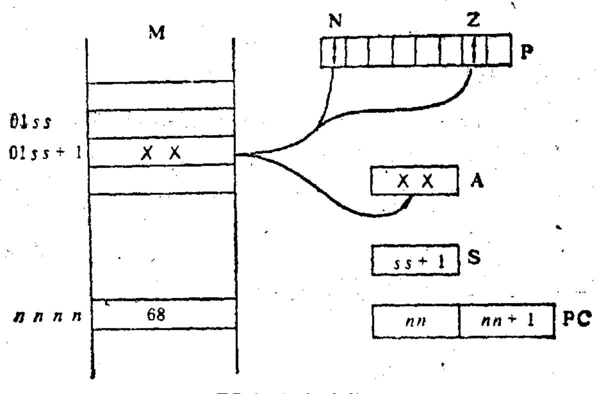


图1—36 PLA指令功能示意图

4. 标志寄存器出栈指令PLP——S+1→S M_S→P

PLP亦为隐含寻址方式的单字节指令，操作码为28。它的功能是先使栈指针自动加1（S+1→S），然后按加过1的栈指针到S所指示的位置取出该单元内容送到标志寄存器P中，PLP指令执行后，标志寄存器P的内容就由出栈的那个单元内容来决定。

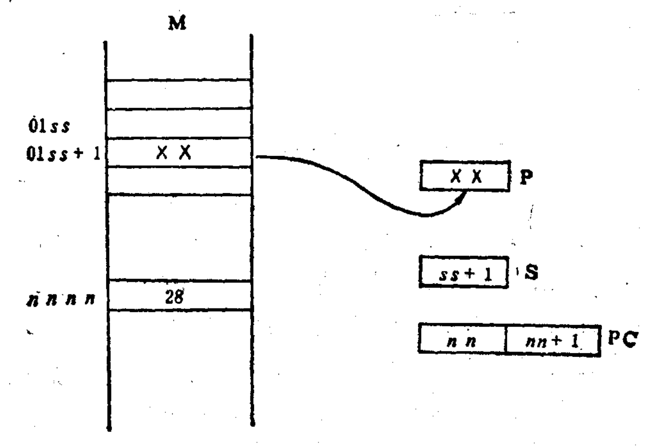


图1—37 PLP指令功能示意图

PLP指令功能如图1--37所示。

5．堆栈指令用法举例

程序在由主程序转入子程序时，若主程序中已经用到的寄存器（指X、Y、A、P）在子程序中也要用到，而且当子程序执行完毕返主之后，主程序还要继续用这些寄存器，为了防止寄存器内容被冲，在转入子程序后就必须首先把这些寄存器状态存入堆栈中保留起来（称为保留现场），然后再执行子程序的主体部分，最后在返回主程序之前还要从堆栈中取出它们以恢复寄存器原内容（称为恢复现场）。保留和恢复现场都是用堆栈指令来实现的，下面我们举例说明保留和恢复现场的具体写法。

```
PHP    ; P进栈，保留P
PHA    ; A进栈，保留A
TXA    ; X→A再进栈，保留X
PHA
TYA    ; Y→A再进栈，保留Y
PHA
 :
 :
PLA    ; 出栈内容由A→Y，恢复Y
TAY
PLA    ; 出栈内容由A→X，恢复X
TAX
PLA    ; A出栈，恢复A
PLP    ; P出栈，恢复P
```

这里除了保留现场时各寄存器进栈的顺序和恢复现场时各寄存器内容出栈的顺序正好相反以外，还应注意6502堆栈指令中只有P和A的进栈及出栈指令，而无X和Y寄存器的堆栈指令，因此当P寄存器和A累加器需要进栈时可以直接使用PHP和PH A指令，而

X和Y寄存器要进栈时必须先用TXA或者TYA 指令把要进栈的内容换到累加器A中，然后再使用PHA 内容使之进栈。出栈情况也一样，出栈的内容欲送入X或Y寄存器时必须通过A累加器中转。

顺便说一句，堆栈在使用的过程中，栈指针S总是指向栈顶的一个空单元，而S的初始值通常是由微型机的监控程序设定在栈底位置01FF处。

七、转移指令

程序在大多数情况下是顺序执行的，即依靠程序计数器PC不断自动加1的操作，指示出下一条指令所在地址，这样计算机就可按照程序中指令的排列顺序一条接着一条的执行下去。但在某些情况下，需要改变程序顺序执行的次序，而转入执行另一个地址中存放的指令，这就要依靠转移指令来实现。

1．无条件转移指令

JMP——转移到操作数字段所指定的地址

JMP指令用于控制程序执行的路线，将程序由通常的顺序执行状态转向另一个指定的目标地址。这个转移的目标地址就存放在JMP指令的操作数字段中。在寻址方式上，JMP 指令只用绝对寻址和间接寻址两种。查一下MPU6502指令表就可知道，6502的间接寻址方式仅仅用于JMP这一条指令。

JMP指令的操作码字节编码如下：

7	6	5	4	3	2	1	0 ←位数
0	1	x	0	1	1	0	0

表1—22所示为两种寻址方式对应的JMP操作码。

以间接寻址方式为例的JMP指令功能前面已有过叙述，请见§1—1图1—1。

由图1—1可见，放在0300～0302单元中的无条件转移指令

表1—22　　　　　　JMP对应的寻址方式

x	指令操作码	寻　址　方　式
0	4C	绝对寻址
1	6C	间接寻址

JMP （$1000)执行后，PC中被送进了由间接地址1001和1000单元的内容构成的转移目标地址码，所以下条指令地址就不是0303而是2FFF。一旦程序转入到这个目标地址2FFF之后，PC就在这个目标地址基础上依次加1使程序又按顺序执行的方式执行下去。

2．条件转移指令

这类指令共有八条，都是采用相对寻址方式的两字节指令，相对寻址方式在§1—1中已作过介绍。各种条件转移指令的功能都是根据指令中要求的条件是否满足来决定是进行分支转移或是继续执行下一条指令，若转移则转移的目标地址用转移步长的形式存放在指令机器码的第二字节中。条件转移指令虽是根据P寄存器中各标志位状态来确定程序是否转移，但它本身的执行却不影响P寄存器状态。条件转移指令的重要性在于它们使计算机有了判断功能，即可以根据具体条件的满足与否来决定执行哪一段程序分支。下面对这八条条件转移指令逐一进行介绍。

(1) BEQ——如果标志位Z＝1则转移，否则继续

它的操作码是F0，这里举一个例子来说明BEQ指令的功能。

```
存贮器地址        指令
   ⋮               ⋮
  030E           LDA  $06
  0310           BEQ  $0342
  0312           AND  #$0F
   ⋮
```

0342 ADC $6000

⋮

执行到BEQ $0342这条指令时，如果前一条指令LDA $06执行结果由06单元取入A的数是八位全0码而使标志位 Z=1，则 BEQ $0342执行之后就转移到0342 处去执行ADC $6000，若是06单元中的数不是0，则LDA $06执行结果Z=0，那么 BEQ $0342执行之后将继续顺序执行下一条指令 AND #$0F，可见程序由于BEQ $0342的执行而产生了两个分支。此例中BEQ $0342的第一个指令字节所在地址是0310，而转移的目标地址是0342，所以BEQ $0342的机器码指令为F0 30两字节，F0是操作码，30是转移步长。BEQ $0342 执行后不论是否转移，都不影响P寄存器中各标志位，而维持它们的原有状态不变。

图1—38所示为BEQ的指令功能。

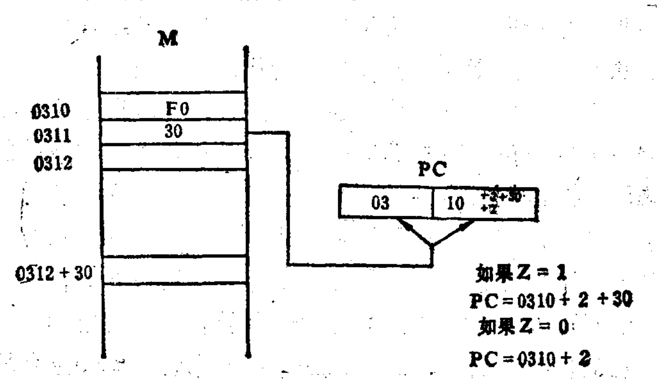


图1—38 采用相对寻址的BEQ功能示意图

(2) BNE——如果标志位Z=0则转移，否则继续

它的操作码是D0，很明显它和BEQ的区别仅在于转移所要求的条件由Z=1改变为 Z=0，把上例程序用 BNE 加以变动后，可以看到，这两段程序完成的功能是相同的。

存贮器地址 指令

```
  ⋮            ⋮
030E          LDA   $06
0310          BNE   $0342
0312          ADC   $6000
  ⋮
0342          AND   #$0F
  ⋮
```

此处BNE　$0342的机器码为D0　30两字节。它的指令功能一目了然，不必再画图说明了。

(3) BCC——如果标志位C＝0则转移，否则继续。BCC的操作码为90。

(4) BCS——如果标志位C＝1则转移，否则继续。BCS的操作码是B0。

(5) BPL——如果标志位N＝0则转移，否则继续。BPL的操作码是10。

(6) BMI——如果标志位N＝1则转移，否则继续。BMI的操作码是30。

(7) BVC——如果标志位V＝0则转移，否则继续。BVC的操作码是50。

(8) BVS——如果标志位V＝1则转移，否则继续。BVS的操作码是70。

这些指令都不影响P寄存器状态。

由于已对BEQ指令作了详细说明，其它的条件转移指令就不必赘述了。

3. 转移到子程序指令JSR和从子程序返回指令RTS

这两条指令简称为转子指令JSR和子程序返回指令RTS。

(1) JSR——转移到操作数字段所指出的子程序入口

转子指令JSR仅使用绝对寻址这一种寻址方式，它的操作码

是20。

转子指令和转移指令的区别在于转移指令控制程序转出之后就不再返回了，而转子指令使程序转向子程序之后，当子程序执行完毕时还要再返回主程序被打断处。实现这个返回是依靠在子程序末尾使用一条子程序返回指令RTS，就可以控制程序自动返回到主程序被打断的地址处(称为返回地址或断点地址)。

例如：

存贮器地址	指令
	⋮
0350	JSR　$6700
0353	⋮
	⋮
6700	PHA
6701	⋮
	⋮
	RTS

当程序执行到0350单元中的转子指令 JSR　$6700时，会自动转到6700单元执行子程序的第一条指令PHA，并接着顺序执行下去，直到子程序结束处的RTS指令被执行后，程序就又回到主程序中刚才被打断的0353处去执行 JSR　$6700的下一条指令。所以JSR指令的操作必须首先把返回地址送到堆栈中保护起来，然后再把JSR指令操作数字段所指出的子程序入口地址送入PC。

JSR　$aabb的指令功能说明如图1—39所示。

由图1—39可见，放在nnnn～nnnn+2单元中的指令 JSR　$aabb的第三字节所在地址码nnnn+2的高字节存进了堆栈中的01ss单元；低字节存进了01ss－1单元，栈指针S最后指向01ss－2单元，随后JSR　$aabb所指明的子程序入口地址aabb 被送进程序计数器PC。至此JSR　$aabb指令执行完毕。

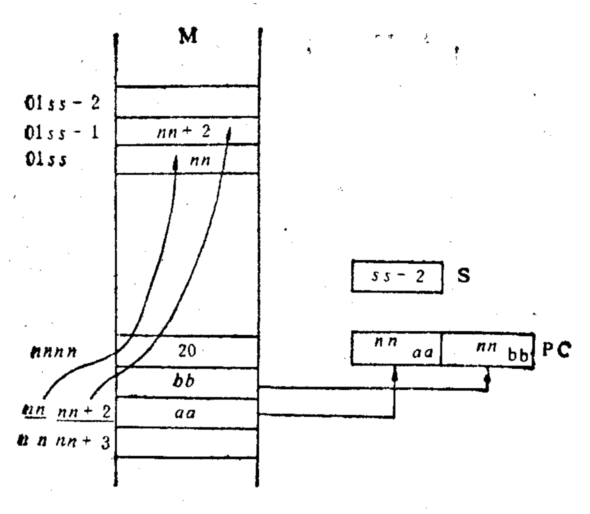


图1—39　JSR　$aabb 指令功能示意图

我们应特别注意，送到堆栈中保存起来的返回地址并不是真正的返回地址nnnn+3，而是JSR ＄aabb 的第三个指令字节所在地址，它比真正的返回地址少1，这个返回地址少1的情况将在子程序末尾处的RTS指令中得到增1修正，MPU6502中JSR指令这样处理返回地址的方法是为了加快JSR指令执行速度。

(2) RTS——由子程序返回到主程序断点处

RTS是采用隐含寻址方式的单字节指令，操作码为60。

由上面的叙述可知RTS指令的作用是和 JSR 指令配合起来实现返回主程序断点处。由于执行JSR 指令时送进堆栈保存的并不是真正的返回地址，而是比返回地址少1，所以在执行RTS指令时就必须把保留在堆栈中的地址取出再作加1修正，就得到了真正的返回地址。把这个返回地址送进PC就实现了返回主程序断点处的操作。这就是RTS指令的操作功能。

RTS的指令功能说明如图1—40所示。

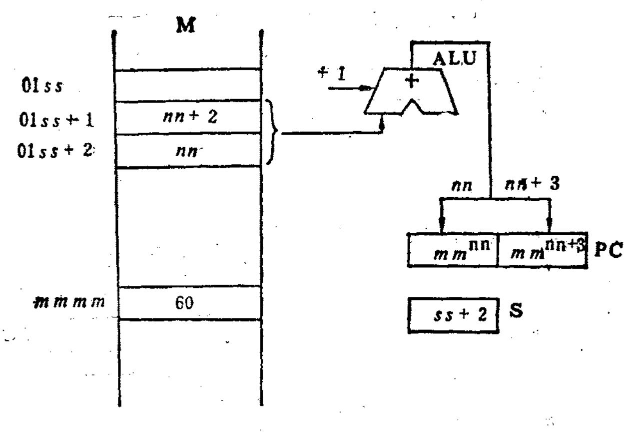


图1—40　RTS　指令功能示意图

JSR和RTS两条指令的执行都不影响标志寄存器P。

4. 中断返回指令RTI

MPU 6502在执行主程序过程中，若遇有外围设备中断请求，则6502在中断响应周期中要将P寄存器及断点地址PC保留进栈，然后才转入为外围设备服务的中断服务程序。等到服务程序执行完毕时也要返回主程序，这个返回是依靠在服务程序末尾处放一条RTI指令来实现。所以RTI指令的操作功能是从堆栈中取出在中断响应周期中保留的P及PC值，送回P和PC中，这就实现了恢复标志状态及返回断点（有关中断的内容将在下面有关章节中介绍）。

RTI是采用隐含寻址方式的单字节指令，它的操作码是40。

RTI的指令功能说明如图1—41所示。

由图可见，放在nnnn单元中的指令RTI执行后，响应中断时保存在堆栈01ss+1单元中的P寄存器内容出栈，送回到P寄存器中，恢复了各标志位原状态。保存在01ss+2单元中的返回地址低字节bb送回PC的低字节，保存在01ss+3单元中的返回地址高

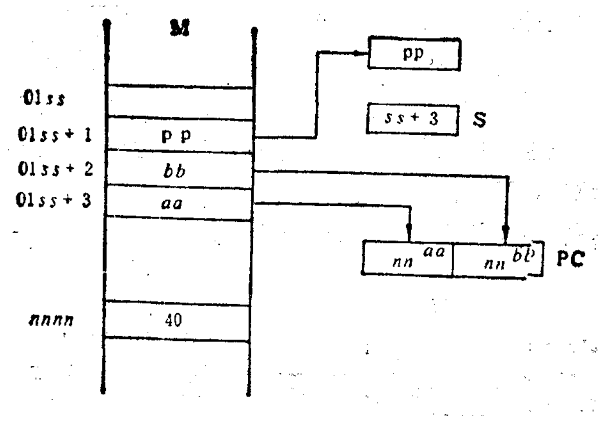


图1—41 RTI指令功能示意图

字节aa送回PC的高字节中，这就实现了返回主程序。栈指针S的内容由ss变成了ss+3。在程序编制中 RTS 使用在子程序结尾处，而RTI则使用在中断服务程序结尾处。由于中断响应转入中断服务程序比起转入子程序要多一个保护P寄存器进栈的操作，所以 RTI 要比RTS多一个P寄存器出栈的操作。而RTS则比RTI多一个返回地址出栈后再做增 1 修正的操作，因此这两条指令虽然都是返回指令，却不能互相代替，在编写程序时切切不能用错了。

5．软件中断指令BRK

BRK是隐含寻址方式的单字节指令，它的操作码是00。这条指令的功能从它要转移到一个目的地址这一点来看类似无条件转移指令的功能，但它转移的目的地址要用中断向量地 址 FFFF 和 FFFE 来指定，而且在转移到目的地址之前要把 P 寄存器内容 及 BRK指令所在地址码加 2(PC+2) 的内容送进堆栈保存。从这点来看，它更类似于中断响应的操作内容，可是这个“中断响应”并不是由送到6502微处理机引腿$\overline{IRQ}$上的外围设备中断请求所引起的，而是由指令BRK所引起的，所以称之为软件中断指令。

执行BRK指令对P寄存器的影响是将B标志位置1，表示 BRK

指令已被执行，此外还使 I 标志位置1，以禁止对外围设备中断请求的响应。所以在执行 BRK 指令时送进堆栈保护的P寄存器状态中 B、I 两标志位是为1的，而其它标志位维持原状态不动。还应注意，被送进堆栈保护的断点地址并不是位于BRK指令的下一条指令地址，而是比它多1。

BRK指令功能说明如图1—42所示。

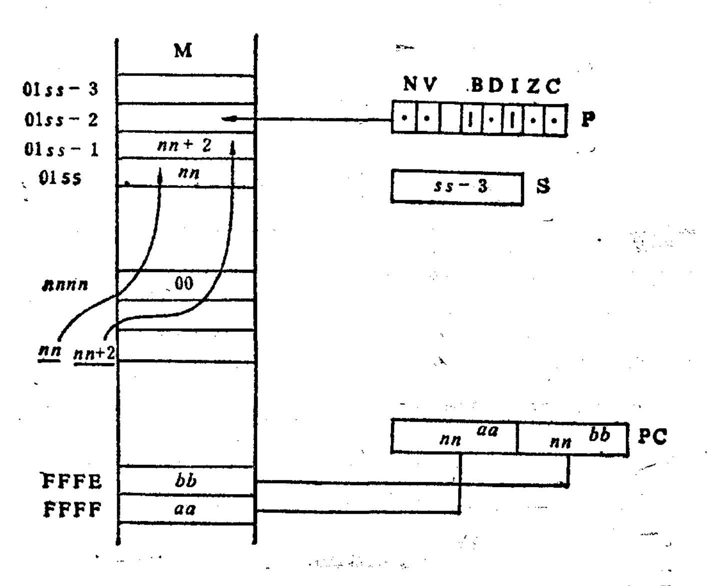


图1—42 BRK 指令功能示意图

由图可见放在nnnn单元中的指令BRK 执行后，断点增1的地址码nnnn+2高字节存进堆栈01ss单元，低字节存进堆栈01ss-1单元，而B、I已被置成1的P寄存器内容存进01ss-2单元，栈指针S的内容由ss 变为ss-3，中断向量 FFFF及FFFE 两单元中存放的中断服务程序入口地址 aabb 分别送进 PC的高字节及低字节，从而使程序转向从aabb开始的程序段。

由于FFFF和FFFE两单元是MPU6502规定的中断向量地址，

同时这两个单元又为BRK指令提供转移的目的地址，所以实际上由FFFF和FFFE两单元所指出的从aabb开始的程序段一开始就是对B标志位查询的程序，如果查到B为1则转入为BRK指令服务的专用程序中（这个专用程序常用作由用户程序返回监控程序的控制）；如果查询到B为0则转入为中断响应服务的中断服务程序中。正是由于这个原因，使得6502对外围设备中断请求的响应时间加长了。

6、转移类指令用法举例

（1）用BRK指令控制由用户程序返回监控程序（APPLE Ⅱ PLUS）

存贮器地址	指令	
0300	CLD	
0301	CLC	
0302	LDA $6000	
0305	ADC $6001	
0308	STA $6002	
030B	BRK	；用BRK指令转向软件中断专用程序
⋮		
⋮		
FA86	STA $45	；A内容⟹45单元保存
FA88	PLA	；栈中保存的P内容出栈⟹A
FA89	PHA	；再送回栈中以恢复栈状态不动
FA8A	ASL A	；把送进A中的P内容左移三次使B标志位送到了A的最高位
FA8B	ASL A	

```
FA8C    ASL A
FA8D    BMI  $FA92       ; B=1转FA92的软件中
                           断专用程序
FA8F    JMP ($03FE)      ; B=0转外设中断服务
                           程序
FA92    PLP              ; 此段为软件中断专用程
                           序，用来显示和打印用
                           户程序断点处各寄存器
FA93    JSR  $FF4C         状态
FA96    PLA
FA97    STA  $3A
FA99    PLA
FA9A    STA  $3B
FA9C    JSR  $F882
FA9F    JSR  $FADA
FAA2    JMP  $FF65       ; 断点处理完毕返回监控
 ⋮       ⋮                 程序入口FF65
FFFE    86               ; 中断向量地址中的内容
                           为BRK，指令转移地址
FFFF    FA                 FA86
```

以上这个程序中 0300～030B 段是用户程序（见§1—2 算逻运算类指令用法举例），它的最后一条是 BRK，因此用户程序执行完最后一条BRK指令就转向中断向量地址FFFF及 FFFE 所指出的地址入口FA86中去，由 FA86 开始的一段程序首先就是把栈中保存的P寄存器内容取到A中再判其中的B标志位是0还是1，若是0就用JMP（$03FE）再转向中断服务程序入口，若是1就转向用BRK设置的软件中断专用程序入口FA92，而由FA92 开始的这段

程序功能是把执行BRK指令时的各寄存器状态显示和打印出来，最后返回计算机的监控程序入口FF65，于是计算机就进入了监控程序控制之下，显示器上显示出 APPLE Ⅱ 监控程序提示符*。

以上列举的程序除了前面一段是用户程序外，由 FA86 开始的程序都是 APPLE Ⅱ 监控程序的有关段落，FF4C、F882、FADA为入口的几个子程序也都在监控程序中，此处不另单列出来。

由以上分析看到，BRK指令可以用来在用户程序中设置断点，使用户程序停顿下来并回到监控程序控制之下，这样就可用监控命令来检查用户程序执行的结果。

此例中除BRK外还有其它几条转移类的指令JMP、JSR、BMI亦被用到了。

(2) 找两个无符号数中的较大数

欲将存贮单元 6000 和 6001 中的较大数放入存贮单元6002，假设存贮单元 6000 和 6001 的内容都是无符号的二进制数。例如 (6000) = 3F，(6001) = 2F 则程序执行后应有 (6002) = 3F。

存贮器地址	指令	
0300	LDA $6000	；取出6000单元中的数⟹A
0303	CMP $6001	；A-(6001)即6000和6001两单元中的数做比较
0306	BCS $030B	；C=1够减，6000单元中数大，转030B
0308	LDA $6001	；C=0不够减，把6001单元中的大数⟹A
030B	STA $6002	；将大数⟹结果单元6002
030E	BRK	；使此程序停顿，返回监控程序

这段程序的思想就是用 CMP 指令来比较两个数，然后利用

BCS 指令进行程序分支，把大的那个数放在累加器A中再送入结果单元，最后使用BRK指令使程序停顿下来。由本例和第二章的程序设计举例中可以看到，在比较指令之后常使用条件转移指令来进行程序分支。

八、空操作指令NOP

这是隐含寻址方式的单字节指令，操作码为EA。执行NOP指令除了将程序计数器 PC+1 及占用了两个时钟周期的执行时间外，其它什么影响也没有。故称为空操作指令。

但是这条指令在程序调试的过程中有用处，例如被调试的程序由于某种原因需要删除掉几个指令字节，那么这几个单元就可以都用NOP来填补，举一个具体例子来说明，在一段包含有转子程序的主程序被调试时，为了便于调试，可先取消转子程序部分的调试，那么就可把主程序中的 JSR 指令所占的三个字节都用NOP指令填补，等主程序调试成功后，再把这三条NOP指令占用的位置重新放上JSR指令。

所以NOP指令在正式的程序中一般不用，而在程序的调试中却常用到。

§1—3 6502的新发展——R65C02

前几年ROCKWELL 公司已经把6502的主频由 1 兆赫提高到2兆赫——即6502A。APPLE Ⅱ微型计算机以及英国的 BBC 微型计算机采用的就是这种主频为 2 兆赫的 6502A。6502A 除了主频提高一倍之外，别的方面与6502完全兼容。去年，ROCKWELL公司推出了更高级的产品——R65C02。它的引腿和软件与6502完全兼容，但是它采用了先进的互补型MOS (CMOS) 工艺，因而使芯片的功耗大大地降低，而主频最高将可达 6 兆赫。该公司宣称，65C02的功能在时钟停止期间最大功耗为 10μW (微瓦)；

时钟工作时，一般情况下平均每兆赫 4mA（即20毫瓦）。这样小的功率消耗将很有利于它应用于携带式计算机和航天技术。

65C02 不仅仅是比6502减少了功率，提高了速度，更重要的是它在指令系统方面除了做到完全包括6502的全部指令之外，又克服了原来6502的某些指令方面的缺点，以及增加了一些新的指令和新的寻址方式。这样就使得指令系统的功能更完善。为了说明65C02 指令系统，我们借这个机会谈谈6502指令系统中的个别问题。这些问题在系统设计（特别是系统软件设计）中是极为重要的。一旦在系统调试中碰到，如果你不了解这些问题，那你就不可能找到系统的故障所在。

这些问题是：中断返回指令，条件跳转的时序，间接寻址方式以及变址寻址方式的总线周期等问题。对于这些问题 65C02对前二者没有进行修改，以便做到与 6502 软件兼容，而对后二者 65C02 则完全改正了出现的问题。

我们首先来谈谈6502的间接寻址方式中可能产生的严重的问题。前面我们已经谈到，间接寻址方式仅仅有一条指令用到，这就是无条件转移指令（JMP）。它是一条三字节指令，它的操作码为6C（十六进制），第二个字节包含一个存贮器地址的低字节，第三个字节则包含着存贮器地址的高字节。如指令JMP（$0300）即可表示为：

6000　6C　00　03

6000（十六进制）是这条指令操作码存放的地址。如果存贮单元0300（十六进制）内存放的是（十六进制）：

0300　00

0301　60

那么这条指令执行结果将是 6000（十六进制）——→PC（程序计数器），这种情况是正常的，这正是我们前面所讲述过的。但是如果上面的指令改为：JMP（$02FF）即：

6000 6C FF 02

02FF 00

0300 60

0200 38

按照6502指令系统的规定这条指令没有任何错误。但是这时既使我们同样在2FF单元放上00，300单元放上60，而执行这条指令却不能把6000送入PC（程序计数器），原因是6502在02FF单元取出00之后，将02FF地址进行加1运算，本应加出0300，但由于6502设计时没有考虑到在这种情况下，应延长一个时钟周期把进位C的值加到高八位地址上去，因此就丢失了进位。结果使得6502下一次将到0200单元中去取高八位地址。这样这条指令执行的结果将是使3800（十六进制数）⟶PC（程序计数器）。为了防止这种情况发生，在使用6502的程序设计中，要记住不要在每一页的末尾使用间接寻址。而65C02就已经考虑到这种情况，在设计上就得到了改正。

6502的第二个比较大的问题是某种特殊情况下的总线读周期问题。我们知道6502芯片对于存贮器的读写控制只有一根引线，即R/W线，而不象别的微处理机（如Z80）那样有读（$\overline{RD}$）、写（$\overline{WR}$）两条控制线，因此在进行跨页变址时就可能发生问题。我们举例说明：如指令LDA ＄6003,X。这条指令如果X值比较小时，需要四个时钟周期。第一个时钟周期6502将取出操作码BD（十六进制）；第二、第三个时钟周期则从存贮器中读取基地址的绝对值即6003（十六进制），当然6502先读的是03，然后才是60，接下去在第四个时钟周期一开始6502内部完成6003与X值相加之后，如果加出的结果不跨页——即不出现进位——的话，6502马上就将这个值送到地址总线，从而使指定存贮单元的值出现在数据总线再送到累加器A。这就是这一条指令的操作过程。但如果在第四个时钟周期一开始，6502内部在完成与X的值相加时出现跨

页——即出现进位——的情况，将自动增加一个时钟周期，这种情况下第四个时钟周期6502内部将再进行一次加法运算，第五个时钟周期才会在地址总线上出现指定的地址，以完成将指定的存贮单元的值读入累加器A。这就是在跨页情况下的绝对变址寻址的操作过程。这个过程中显然在第四个时钟周期实际上并不对存贮器进行操作，完全是6502内部进行变址计算。但是这个周期的ϕ_2期间，读写控制线（R/W）仍然为高电平，因此这时仍然要根据地址总线上出现的地址进行读操作，而指定地址只有在第五个周期才会出现在地址总线上。因此第四周期进行的读操作并不是对指定的地址，而是对某个特殊的地址，但由于进行的是读操作，一般情况下这将毫无影响。因为这既不破坏该存贮单元的内容，也不会被送入6502的寄存器，所以不会影响程序的正常运行。但是在某些应用中，如果采用了某些可编程大规模集成电路的芯片，如6850、6522、6520等，这些芯片的中断标志寄存器往往是以对某个地址进行一次读或写操作来进行清零。如果遇到这种情况，系统将发生很大的问题，将会造成错误的结果。这是我们在系统设计中必须注意的问题。不过65C02 改进了设计，这种情况下的第四个周期地址总线将会自动脱开接口芯片，从而避免了出现这种问题。

6502的另外两个问题不象上面两个问题那样严重，甚至可以说算不上什么问题，我们把它提出来不过是为了引起大家在软件设计中应予以注意。第一个问题是关于中断返回指令(RTI)和子程序返回指令（RTS）方面的问题。上面已经谈过，从功能上讲RTI返回指令相当于PLP加上RTS指令。但是要注意对6502来讲，实际情况并非如此。因为在中断响应过程中推入堆栈保存的是下一条本来要执行的指令地址，这正是断点地址，而在转子指令（JSR）中推入堆栈保存的并不是断点地址，而是JSR指令的第三个字节的地址。因此RTS指令执行过程中必须对此进行修正再将

它们送程序计数器PC，以实现返主，了解它们的差别对于编写实现跟踪和单步功能的软件很重要，另一个问题是条件转移指令的时序问题，这一类指令执行时间可能是2～4个时钟周期。第一个时钟周期用来取操作码，第二个时钟周期用来取指令的第二个字节，这里表示的是相对跳转的偏移量。如果跳转条件不具备，则下一个时钟周期将是取下一条指令。如果跳转条件成立，下一个周期将用来把偏移量与程序计数器的低八位相加，若出现进位或借位，第四个时钟周期将用来修正程序计数器的高八位。因此条件跳转指令的执行时间是：如果不跳转只要两个时钟周期，如果跳转但只是在存贮器本页之内实行跳转则需要三个时钟周期，如果是在存贮器地址跨页之间实行跳转则要花四个时钟周期，这些问题在计算延迟时间间隔时必须予以注意。

对于这里讲的后面两点，65C02 没有进行更改，仍然保留了下来，以便它完全可以执行6502的软件。

65C02 增加了一些新的指令，使其功能较标准的6502更趋完善。新增的指令主要是：按零页地址中任意一位的状态而进行条件跳转的指令；无条件相对跳转指令；X、Y寄存器的栈操作指令，零页地址中任何一位置 1 和清零的指令；存贮器中任何一个地址单元清零指令；位测试并逻辑与和位测试并逻辑或的指令。以下我们分别简要地介绍一下：

BBR_x(按位为零跳转）指令：这条指令可以使零页地址中的任何一位作为一个标志位，当零页中某个地址单元中的指令位为零时即按指令所给出的偏移量进行跳转。这条指令长度有三个字节，头一个字节是操作码，第二个字节是做为标志位所在单元的零页地址，第三个字节则给出了跳转的偏移量。具体是哪一位由操作码中的Bit 6到4二进制数所决定。

BBS_x(按位为 1 跳转）指令：这一条指令同上一条很类似，不同的就是这里跳转的条件不是 0 而是 1。

BRA（无条件相对跳转）指令：6502没有无条件相对跳转指令，而只有无条件绝对跳转（JMP）指令，这是一个三字节的指令。65C02加入这条指令，可以在跳转的距离不长的情况下使用，而它只有两个字节。

65C02增加了四条有关X、Y进栈和出栈的指令。在6502中X、Y不能直接进行栈操作，需要它们进栈时，首先要把它传送给累加器A，然后再用A进栈指令，即TXA和PHA（或TYA和PHA）。在出栈时，同样要先将栈中内容弹出到累加器A，然后再将A的内容传送到X或Y寄存器，即PLA和TAX（或PLA和TAY）。这样的进栈和出栈既费时间又费内存空间，65C02增加四条指令可以使X、Y象累加器A那样直接进栈和出栈。

RMB_x（零页位清零）指令和SMB_x（零页位置1）指令，这两条指令可以分别使存贮器零页中某个地址单元中的任何一位清零或置1，而不影响65C02内部任何寄存器的值。

STZ（存贮单元清零）指令：这条指令将使存贮器中任何一个地址单元清零而不影响任何内部寄存器的值。它允许四种寻址方式：零页寻址、绝对寻址、X零页变址和X绝对变址。

TRB（位测试并逻辑与）指令：这条指令是6502的BIT指令和AND指令的组合。BIT指令执行中将有关的存贮单元的Bit7和Bit6送标志寄存器(P)的N位和V位。然后是累加器A的值同有关存贮单元的值进行逻辑与，结果送回该存贮单元（不影响A的值），标志位Z的值将由结果而定。

TSB（位测试并逻辑或）指令：这条指令是6502的BIT指令和OR指令的组合。它和上一条除了逻辑或代替逻辑与以外其余相同。

此外，65C02除了有6502的十三种寻址方式之外，还增加了一种新的寻址方式。这就是零页间接寻址方式，这种寻址方式中的两个字节地址存贮在零页存贮单元中。6502只有JMP指令有间

接寻址方式，现在65C02对一些主要的指令如ADC、AND、CMP、EOR、LDA、ORA、SBC、STA等都可以使用零页间接寻址。

另外，对一些6502原有的指令（如BIT、INC、DEC等）允许它们使用更多的寻址方式，使这些原有的指令有更强的功能，这里就不一一赘述了。

我们一再提到的65C02不过是CMOS 6502家系中的一员。ROCKWELL公司在这个家系中还将会推出更多更新的芯片，现在已知的芯片有65C102、65C112等。这些请读者参阅有关的最新资料。

Retro Workshop
果粉工作室

第二章　6502汇编语言程序设计

汇编语言是一种和计算机类型及其指令系统密切相关的程序设计语言。这是汇编语言与高级语言不同之处。例如以6502为微处理机的6502汇编语言和以Z80为微处理机的Z80汇编语言就有很大差异，两者不能通用。由于汇编语言紧密依赖于具体计算机的结构和指令系统，因而程序员必须对具体计算机的硬件结构、寻址方式、指令系统及伪指令有很仔细的了解，然后才能设计汇编语言程序。以上已经介绍了6502的内部结构；6502的指令系统及寻址方式，在这个基础上本章将继续介绍6502汇编语言的语句格式，伪指令及6502汇编语言程序设计的基本方法。

§2—1　汇编语言源程序的语句格式

用6502汇编语言所编写的源程序是由两类语句构成的。一类是指令语句（即由指令系统中的指令构成的语句）；另一类是伪指令语句。这两类语句都必须严格遵守语句格式的规定，即每一个语句最多可由四个字段组成：

标号	操作码	操作数	；注释
Label	Opcode	Operand	Comment

字段之间用空格作为分隔符，但同一字段内不要使用空格，一般说来，对于每一个语句操作码和操作数这两个字段是必不可少的，而标号和注释则视具体情况而定，也就是说，这两个字段是可选字段，并不是每一个语句都必不可少的。

下面举一个例子，说明由四个字段构成一个语句的方法。

标号	操作码	操作数	注释
START	LDA	$6000	；将6000单元的内容送到A累加器
	STA	$0350	；将A内容送到 0350 单元

注意：在把源程序送入计算机时，注释字段应用英语书写。

关于这四个字段的用法，详细说明如下：

一、标号

标号是一条汇编语言指令所在存贮单元的符号表示，即标号表示的是一个地址码。当一条指令有标号时，在汇编过程中，汇编程序将把存放该条指令目标码第一个字节的存贮单元地址值赋给这个标号。因此在转换成目标程序时，标号就被具体的存贮单元地址所取代了。对于子程序入口、转移目标、计数器单元、常数单元或工作单元都可使用标号。例如：

```
        LDA    #0        ；和=0
        TAX              ；变址计数=0
SUMD    CLC              ；清进位
        ADC    $6000,X   ；和=和+数据
        INX              ；变址计数+1
        CPX    #10       ；十个数是否加完
        BNE    SUMD      ；未加完，继续求和
        STA    $0350     ；加完，送结果
        BRK
```

这是一个求和程序，对从6000单元开始的十个数求累加和，结果存入0350单元，并假设和数不大于一个 8 位二进制数，这样可以不考虑进位。

这个程序中使用了一个标号 SUMD，当程序执行到BNE SUMD 时，若转移条件满足，下一条待执行的指令就是同标号

SUMD相对应的存贮单元中的那一条CLC。

可见使用了标号，程序员就不必算出程序中存贮单元的具体地址值，而且使得程序单元容易查找和修改。一旦对某条指令使用了标号，那么这个标号就可以作为地址或数据用于其它指令的地址段中。

使用标号应当注意以下几点：

1．标号必须以英文字母 A～Z 开头，APPLE Ⅱ 规定标号以八个字符为限，以空格符作为结束。

2．不得使用操作码助记符作为标号。

3．不得使用A，X，Y，P，S这几个6502中程序员可访问的寄存器名字作为标号。

二、操作码

操作码由指令助记符或伪指令构成。

6502助记符一律是用三个字母表示，而伪指令在 APPLE Ⅱ 中由二个以上字母组成（关于伪指令在§3—2中将作介绍）。

例如：

```
ORG    $300     ；程序起始地址是300单元。
LDA    #$FF     ；取立即数$FF到A。
STA    $10      ；再存到10号单元中。
BRK
```

在这个小程序中ORG、LDA、STA、BRK都是操作码，其中ORG是伪指令，其余是指令助记符。操作码是每一个语句的必选字段。

三、操作数

操作数字段中放的是操作码所要操作的数据。操作数出现的形式应遵从6502寻址方式的规定，这在第一章中已做过介绍，在汇编语言中操作数还可以使用标号、常数、表达式，分别说明如下：

1. 标号

在前面介绍标号字段时所举的对十个数据求累加和的程序中BNE SUMD这条指令语句中的操作数字段就是以标号 SUMD 的形式出现的。用定义了的标号作为操作数这是经常要用到的。

2. 常数

仍然以对十个数据求累加和的程序为例，可以看到其中LDA #0 CPX #10以及STA $0350 这几条指令中的操作数都是以常数形式出现的。

使用常数时应当注意又有十进制常数、十六进制常数、八进制常数和字符串常数之分。

APPLE Ⅱ规定十六进制常数前面必须加$字符。例如 STA $0350中的0350是个十六进制数。

十进制常数前面则不加字符。例如CPX #10中的10是个十进制数。

八进制常数前面加@字符。微型计算机中一般不使用八进制。

字符串常数用跟在单引号之后的字符来表示 ASCII字符。例如LDA #'A表示的是把字母A所对应的ASCII码$C1（APPLE Ⅱ）送入累加器A。

当所表示的ASCII 字符不止一个时，在最末一个字符之后应再缀有尾引号。例如伪指令语句DFB 'ABCD'，'XYZ'表示的是把字符串ABCD及XYZ所对应的 ASCII 码顺序送到由此条伪指令所在地址单元开始的七个单元中去，字符串之间用逗号隔开。

3. 表达式

操作数可用表达式的形式出现。表达式由运算符+、-、×、/、>、<构成，例如 LDA SECD-1 此条指令的操作数使用了标号和常数组成的表达式，表示的是把标号SECD 对应的地址单元数减 1 作为有效地址，从中取出操作数送到累加器A。

运算符<是表示选择高字节，运算符>是表示选择低字节。

例如

```
LDA  #<10000  ; 将10000的高字节送入A
STA  T1CH     ; 再送入T1CH单元
LDA  #>10000  ; 将10000的低字节送入A
STA  T1CL     ; 再送入T1CL单元
  ⋮
```

十进制数 10000 必须用两个字节来放置。此处<和>运算符就是用来选择高、低字节的。

在一个表达式中有多个运算符时，对表达式进行运算的规则是按着从左到右的次序逐项进行计算。即所有的运算符具有同等的优先权。例如表达式5＋<COUNT－2的值是先将5加上10000的高字节数，然后再减去2。

四、注释

注释字段是一个可选字段，并不是每个语句都必须有。注释是用来说明被注释的这条指令的作用以使明瞭程序当前正在做什么。注释必须以分号；开始。

注释的作用仅仅是为了便于阅读，它在汇编时不影响目标程序。

§2-2 伪指令

在 §2—1 中曾经提到汇编语言语句有两类，一类是指令语句，另一类是伪指令语句。伪指令是指不直接被翻译成机器语言的指令，它的作用仅是对汇编程序的一种命令，用来命令汇编程序在汇编时执行一些诸如为程序分配一定的存贮区域，给符号赋值，把给定数据放入存贮单元，留出存贮数据用的存贮区等操作。

伪指令构成的语句写在源程序中，同样可以分为标号、操作码、操作数、注释四个字段。它和指令不同的是一条伪指令不能找到

一条机器指令与之对应，即不能翻译成机器指令，但对程序员来讲则可不必考虑两者的区别，都可同等使用它们来编写源程序。

对于采用6502微处理机的各种型号的微型机来说，它们的指令系统都是相同的，但伪指令却因各个汇编程序而异。例如APPLE Ⅱ、AIM65都用6502微处理机构成，但两个汇编语言所使用的伪指令却不相同，下面介绍的是APPLE Ⅱ 6502汇编语言伪指令。

1. ORG (Origin) ——初始地址伪指令

ORG伪指令在使用时其操作数字段应放置存贮器地址值，用来告诉汇编程序，在这条ORG指令之后的一段程序或数据它们的起始地址是多少。

例如在§2—1操作码这部分介绍的小程序中

```
ORG    $300
LDA    #$FF
STA    $10
BRK
```

使用了ORG $300这条伪指令，汇编程序就会把后跟的由三条指令组成的小程序目标码放在起始地址为 $0300 的一段存贮区域中。可见ORG指令总是写在程序或数据块的开始处。

2. EQU (Equate) ——等值伪指令

EQU伪指令用来把一个常数或一个由已赋值符号构成的表达式的值赋给另一个符号。例如在§2—1表达式这部分所介绍的小程序，可以用EQU伪指令改写为：

```
COUNT    EQU    10000      ; COUNT = 10000
         LDA    #<COUNT    ; 将10000的高字节送入A
         STA    T1CH       ; 再送入T1CH单元
         LDA    #>COUNT    ; 将10000的低字节送入A
         STA    T1CL       ; 再送入T1CL单元
```

⋮

此处再举一个例子说明EQU伪指令的用法

```
GETNSP   EQU   $F634     ；将符号地址 GETNSP赋
                          以$F634
         LDA   #03
         STA   $3D
         JSR   GETNSP    ；转到子程序入口处
                          $F634
         ASL   A
               ⋮
```

在写程序时，EQU伪指令也应放在程序的开始处。

3. DFB (Define Byte) ——字节定义伪指令

DFB伪指令用来告诉汇编程序在目标代码中留出一个或一串字节位置，并在其中填入DFB伪指令操作数字段所列出的数值。当这些数值是字符的ASCII码时，必须将字符括在单引号'内；当操作数字段是多个数值时，彼此间要用逗号，隔开。例如

```
         ORG   $F9C0
MNEML    DFB   $1C, $8A, $1C, $23, $5D,
               $8B, 'A, 'B, 'C
         DFB   $1B, $A1, $9D, $8A, $1D,
               $23
```

伪指令ORG $F9C0 规定了标号MNEML的地址值是 $F9C0，两条伪指令DFB规定把位于操作数字段的一共十五个数顺序送入起始地址为 $F9C0 的十五个字节中，其中第七，八、九三个地址中存放的是字母A、B、C对应的ASCII码。

4. DW (Define Word) ——数据字定义伪指令

DW伪指令用来告诉汇编程序在目标代码中留出一个或若干个数据字位置，并在其中填入DW伪指令操作数字段所列出的数

值。一个数据字包括两个字节，其中填入所列数值时应遵循的规则是：先填所列数据低字节，后填高字节。当操作数字段有两个操作数时，彼此间要用逗号，隔开。例如：

ADDRESS　DW　$C804，　$E4B9

此处的这条伪指令是向汇编程序指明，在ADDRESS 地址单元中存入十六进制数04，而在ADDRESS+1地址单元中存入十六进制数C8，同理在 ADDRESS+2 单元中存入十六进制数 B9，而在ADDRESS+3单元中存入十六进制数E4。由于这样规定了存放顺序，用DW伪指令来存放地址码是很方便的。举例来说，当程序中某处欲转移到$C804地址时，由于已在ADDRESS及ADDRESS+1处定义了数据字，所以只要写一条JMP（ADDRESS）程序即可转移到$C804。

5．DDB（Define Double Byte）——双字节定义伪指令

此条伪指令含义和DW 相同，区别仅在于向留出的两个字节中填入所列数值时，顺序不同。对DDB伪指令来说，是先填所列数值的高字节，后填低字节，这正好和DW相反。例如：

ADDRESS　DDB　$C804，$E4B9

则在ADDRESS单元中存入十六进制数C8，在ADDRESS+1单元中存入十六进制数04，同理在ADDRESS+2单元存入十六进制数E4，在ADDRESS+3单元中存入B9。

6．DS（Define Storage）——存贮区定义伪指令

DS伪指令是用来告诉汇编程序，由标号所指定的单元开始，保留一定数量的内存单元，以供程序使用（例如，将保留的这些内存单元作为中间数据暂存区使用）。需保留的内存单元数目写在DS伪指令的操作数字段。例如：

BUFFER　DS　20

这就是由BUFFER单元开始空出廿个单元。

关于APPLE Ⅱ 的伪指令，本节中只介绍了最常用到的六条，

其余请阅读APPLE Ⅱ有关资料。

本章下面各节将介绍如何用汇编语言编写程序，我们将用具体例子来说明这些程序及其编写方法。为了初学者的方便，在介绍中对每个程序尽量按照目的、示范题、框图、程序的顺序来进行介绍。

§2—3 循环及分支程序

在程序设计中，除了极简单的问题外，一般都要遇到分支和循环的情况，即根据不同条件使程序转向不同的分支，这就是分支程序，或是程序中的某一段要被反复执行许多次，这就是循环程序。

循环程序的结构具有以下几个部分：

1. 初始化部分，它建立计数器，地址寄存器（指针）以及其它变量的起始值。

2. 处理部分，在这部分进行实际的数据处理，这是循环程序中要求重复执行的程序段部分。

3. 循环控制部分，它为下一轮循环做好修正计数器和指针的准备。

4. 结束部分，它分析和存放结果。

注意：计算机对第1部分和第4部分只执行一次，而对第2部分和第3部分则可执行多次。因此循环程序执行时间主要取决于第2部分和第3部分的执行时间。

循环程序可以用来处理数据块。为完成此项任务，程序必须在每一轮处理之后将变址寄存器的内容加1。使它指向数据块中的下一个元素。这样就将使下一轮对下一个存贮单元的数据执行同样的操作。因而计算机可用相同的指令组处理256以内长度的数据块（因为变址寄存器是8位字长）。6502微处理机处理数据块的

关键是采用变址寻址方式。因为变址寻址方式允许用改变变址寄存器内容的方法来改变存贮器的实际地址。

此外，我们从下面所举实例中可以看到在循环程序中已包含了程序分支的概念，每循环执行一次之后是继续重新循环呢？还是不再循环而往下执行新的程序段呢?这里就存在着程序的分支。所以根据某些条件来决定程序语句的执行路线,这就是程序分支。一般地讲，循环程序中的循环处理部分在程序执行过程中将被执行若干次。而对一些单纯的分支程序，则其中每条指令可能只执行一次而不进行循环。各种微处理机都为分支程序设计提供了方便的指令。6502指令系统中的比较指令及条件转移指令就为设计分支程序和循环程序提供了方便。

在以下所介绍的程序中，使用了个别 0 页地址，由于 0 页寻址指令的字节短，执行速度快，所以应尽量多使用。但是APPLE Ⅱ 的 0 页地址绝大部分分配给系统程序使用，用户可以使用的就仅有＄06～＄09，＄19～＄1F，＄FA～＄FD 这几个单元了。

例1 求数据的累加和

目的：计算一串二进制数的和。这串数的长度放在存贮单元6000，而这串数是从存贮单元6001开始放的。将和数存入存贮单元06。假设和数是一个8位数,这样可以不考虑进位问题。

示范题： (6000) = 03

(6001) = 28

(6002) = 55

(6003) = 26

结果 (0006) = (6001) + (6002) + (6003)

= 28 + 55 + 26 = A3

框图：

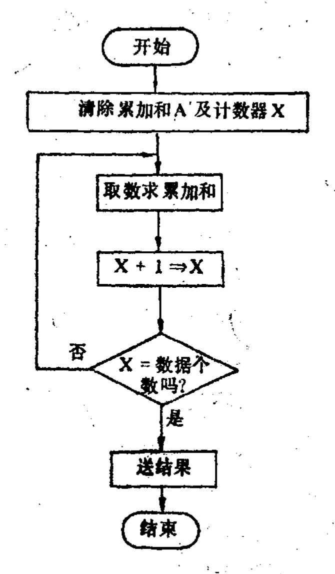


图2—1　求数据累加和程序框图

源程序：

```
        ORG   $0300     ；程序起始地址
        LDA   #00       ；和初值＝0
        TAX             ；计数器初值＝0
        CLD             ；二进制求和，清标志D
SUMD    CLC             ；清标志C
        ADC   $6001，X；和＝和＋数据
        INX             ；计数器＋1
        CPX   $6000     ；次数够了吗？
        BNE   SUMD      ；不够，继续循环
        STA   $06       ；够了，送结果
        BRK             ；结束
```

源程序也可如下编排：

```
        ORG     $0300
        LDA     #00
        LDX     $6000
        CLD
SUMD    CLC
        ADC     $6000, X
        DEX
        BNE     SUMD
        STA     $06
        BRK
```

这一程序两种写法的区别在于，第一种写法求和时是从前面往后面加的，而第二种写法求和时是从后面往前面加。应当注意在这两种写法中，指令ADC操作数部分的基地址用的不同，这是编制程序时应细致注意的地方。另外读者还可以考虑怎样改进它们，使其完成得更快一些。

可以看出，这一程序和§2—1标号部分所举十个数求累加和的程序结构是相同的，区别仅在于此处并不局限于固定的十个数求和，更具有一般性，使编制的程序具有通用性和灵活性，这也是程序编制中应加以注意的。

例2　找最大值

目的：找出一个数据块中的最大值。数据块的长度放在存贮单元6000中，而数据块本身是从存贮单元6001开始存放，将最大值放在存贮单元06中。假设数据块中的数都是8位不带符号的二进制数。

示范题：　(6000)＝05

　　　　　(6001)＝67

　　　　　(6002)＝79

　　　　　(6003)＝15

(6004) = E3

(6005) = 72

结果(06) = E3

因为它是这五个无符号数的最大值。

框图：

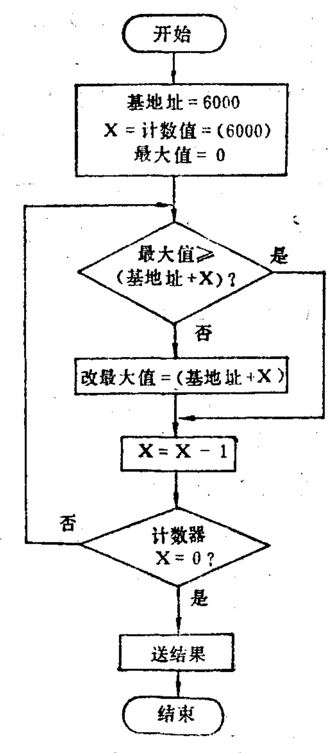


图2—2 找最大值程序框图

源程序：

```
        ORG  $0300
        LDX  $6000        ; 置计数器初值为数的个数
        LDA  #00          ; 最大值 = 0(最小可能值)
```

```
MAXM   CMP $6000, X    ; 下一个数比最大值大吗?
       BCS NOCHG       ; 否，维持A中最大值不变
       LDA $6000, X    ; 是，用该数替代最大值
NOCHG  DEX
       BNE MAXM        ; 继续循环直至所有的数
                       ; 都被比较完毕
       STA $06         ; 存最大值
       BRK
```

可以看到，此程序找最大值的办法是先设一个最大值 0 放在累加器A中，然后由数据块的最后一个数比较起，每次把比较结果中为大的那个数放在A中，循环次数由数据块长度来决定。其实此程序中最大值的初值不设为 0 而设为数据块中被比较的第一个数也是可行的。

程序中使用了BCS指令，由于6502规定作减法时C = 1表示够减，C = 0表示不够减，所以BCS 指令的控制是在够减时（A≥下一个数）跳转，在不够减时（A<下一个数）顺序执行。

例3 确定负数的个数

目的：确定一个数据块中负数（最高位为1)的个数。数据块的长度放在存贮单元6000，而数据块本身从存贮单元6001开始存放。负数的个数放在存贮单元06。

示范题：(6000) = 06
(6001) = 68
(6002) = F2
(6003) = 87
(6004) = 30
(6005) = 59
(6006) = 2A
结果(06) = 02

因为6002和6003单元中为负数

框图：

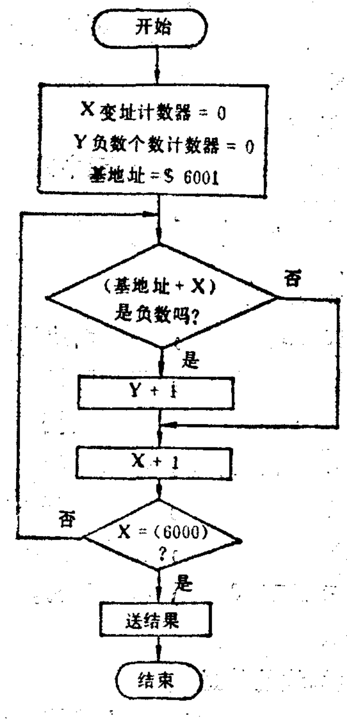


图2—3 求负数个数程序框图

源程序：

```
       ORG  $0300
       LDX  #00       ；变址计数器X＝0
       LDY  #00       ；负数计数器Y＝0
SRNEG  LDA  $6001,X；取一个数
       BPL  CHCNT     ；是负数吗？不是则转
                        CHCNT
       INY            ；是，负数个数加1
```

```
CHCNT INX            ; 变址计数器+1
      CPX  $6000     ; 所有的数都检查完了吗?
      BNE  SRNEG     ; 否，继续检查
      STY  $06       ; 是，存负数的个数
      BRK
```

例4 数据块搬家

目的：将存在6001～60FF地址单元中的数据块按相反顺序传送到7001～70FF地址单元中去。

源程序：

```
      ORG  $0300
      LDX  #01        ; 源数据块计数器初值
      LDY  #$FF       ; 目的数据块计数器初值
NEXT  LDA  $6000,X    ; 取源数据
      STA  $7000, Y   ; 送入目的地址
      INX             ; 源计数器+1
      DEY             ; 目的计数器-1
      BNE  NEXT       ; 次数不够，循环
      BRK
```

例5 三个数据块的传送

目的：将存在6000～6063单元中的第一个数据块送到7000～7063单元中，将存在6100～6163单元中的第二个数据块传送到7100～7163单元中，将存在6200～6263单元中的第三个数据块传送到7200～7263单元中。

源程序：

```
      ORG  $1A
      DFB  $00, $60, $00, $61, $00, $62;将三
                个源数据块首地址填进1A～1F单元
      ORG  $FA
```

```
        DFB $00, $70, $00, $71, $00, $72;将三
                个目的块首地址填进FA~FF单元
        ORG  $0300
        LDX #00          ; 变址计数器X置初值0
LOOP1   LDY #$64         ; 数据块长度⇒Y计数器
LOOP2   LDA ($1A, X)     ; 传送一个源数据到目的地址中
        STA ($FA, X)     ; 第一次是将6000单元内容⇒
                           7000单元
        INC $1A, X       ; 修改源地址，使之增1，第一
                           次是使(1A)=00+1
        INC $FA, X       ; 修改目的地址，使之增1,第一
                           次是使(FA)=00+1
        DEY              ; 一个数据块传送完了吗?
        BNE LOOP2        ; 一个数据块未传送完毕，返回
        CPX #$04         ; 三个数据块都传送完了吗?
        BEQ DONE         ; 是，转结束
        INX              ; 否，将X增2，为取下一个数据
                           块作准备
        INX
        JMP LOOP1        ; 转入下一个数据块传送
DONE    BRK
```

由本例可以看到先变址(X)间接寻址方式的使用方法。本例中的LDA及STA两条指令使用了这种寻址方式，它使得多个数据块的同时处理变得简单、容易。这也是MPU6502独特的地方。

例6 延时程序

目的：由执行一段程序来形成一定的延时时间。因为执行一条指令以及一个循环是要一定时间的，所以可以用控制循环次数的办法来达到一定延时时间的目的。

框图：

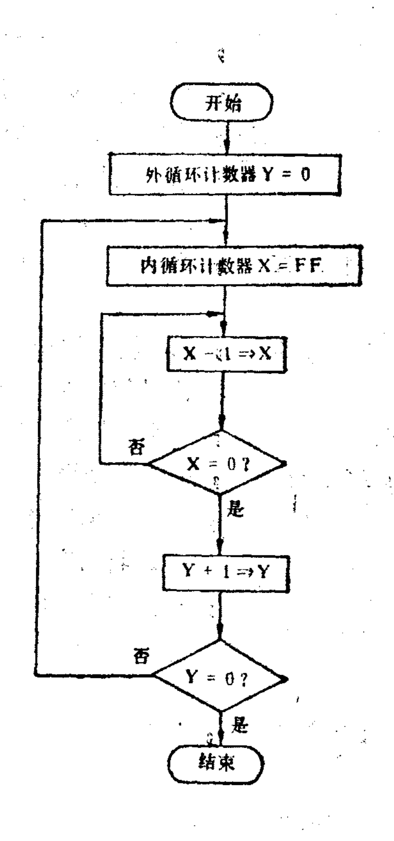


图2—4　延时程序框图

源程序：

```
ORG  $0300
LDY  #00      ；外循环计数器初值
```

```
LOOP1  LDX #$FF     ; 内循环计数器初值
LOOP2  DEX          ; 内循环计数器X-1，为0否
       BNE LOOP2    ; 不为0继续内循环
       INY          ; X为0则外循环次数Y+1
       BNE LOOP1    ; Y不为0继续外循环
       BRK          ; Y为0，结束
```

现在来计算一下这个延时程序所形成的延时时间是多少？查指令表可知，此程序中各条指令的执行时间周期数是

```
       LDY #00      ; 2
LOOP1  LDX #$FF     ; 2
LOOP2  DEX          ; 2
       BNE LOOP2    ; 3，不跳转时为2
       INY          ; 2
       BNE LOOP1    ; 3，不跳转时为2
       BRK          ; 7
```

由于6502的1周期时间是1μs，所以延时时间为：

$$(255(3+2)-1+2+2+3)256-1+2+7$$
$$=327944\mu s \doteq 328ms \doteq 0.3s$$

下面我们再介绍一个实用的延时子程序：

```
WAIT   SEC          ; 2周期
WAIT2  PHA          ; 3周期
WAIT3  SBC #01      ; 2周期
       BNE WAIT3    ; 3周期，不跳转时为2
       PLA          ; 4周期
       SBC #01      ; 2周期
       BNE WAIT2    ; 3周期，不跳转时为2
       RTS          ; 6周期
```

它的程序框图为：

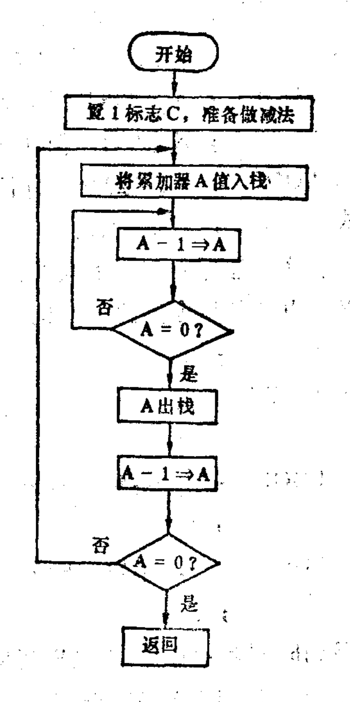


图2—5 WAIT延时子程序框图

由计算可知，WAIT延时子程序的延时时间为：

$\frac{1}{2}(5A^2+27A+26)\mu s$ （A为累加器的内容）

根据对延时时间的需要，程序员事先设定好累加器A的值，然后再转入WAIT子程序。

以上这两个延时程序，都是双重循环程序。对于较为复杂的一些问题，单重循环已经不够用了，而常常需要采用多重循环结构。

例7 8位无符号数排序

目的：将一无符号二进制数数组按升序重新排列。数组的长

度N放在存贮单元6000中，而数组本身从存贮单元6001开始存放。

示范题：(6000) = 06

(6001) = 2A

(6002) = B5

(6003) = 60

(6004) = 3F

(6005) = D1

(6006) = 19

结果：(6001) = 19

(6002) = 2A

(6003) = 3F

(6004) = 60

(6005) = B5

(6006) = D1

方法：采用冒泡法，即从第一个数开始，依次把它与数组中其余N－1个数相比较，并把其中较小者放在第一个单元中。这样，当把第一个数与所有其余的数比较完了以后，在第一个单元中就是数组中最小的数。接着进行第二轮比较，再从第二个单元开始，把第二个数与它后面的其余N－2个数依次进行比较，每比较一次都把较小者放在第二个单元中，当第二轮比较完了之后，在第二个单元中就是数组中次小的数，依此类推，当比较完N－1轮以后，这个数组就已经按升序重新排好，放在原有的N个单元中了。

例如示范题中的6个数

第一轮比较完之后的顺序是：19 B5 60 3F D1 2A

第二轮比较完之后的顺序是：19 2A B5 60 D1 3F

第三轮比较完之后的顺序是：19 2A 3F B5 D1 60

第四轮比较完之后的顺序是：19 2A 3F 60 D1 B5

第五轮比较完之后的顺序是：19 2A 3F 60 B5 D1

框图：

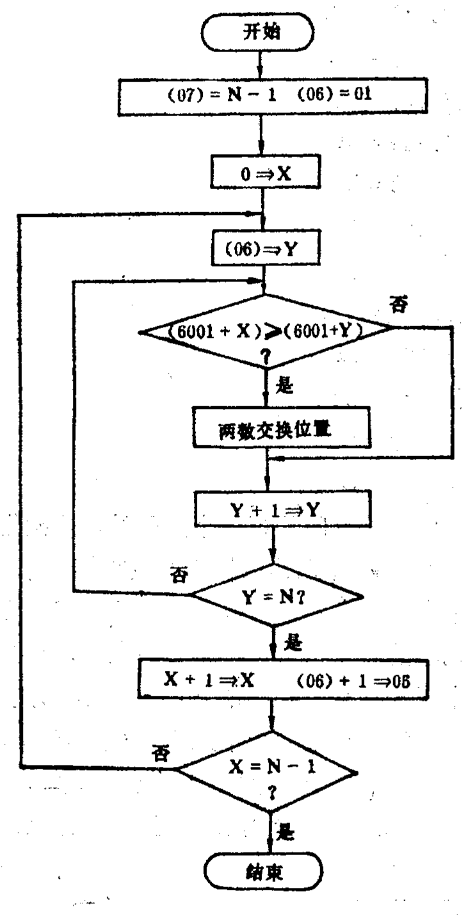


图2—6 排序程序框图

源程序：

```
ORG $0300
LDA $6000
```

```
        STA  $07
        DEC  $07        ; (07)=N-1
        LDA  #01
        STA  $06        ; (06)=01
        LDX  #00        ; X=00
LOOP1   LDY  $06
LOOP2   LDA  $6001,X    ; 前一数与后面各数分别比较
        CMP  $6001,Y    ; 哪个数大?
        BCC  COUNT      ; 前一数小，转COUNT
        PHA             ; 前一数大，交换位置
        LDA  $6001,Y
        STA  $6001,X
        PLA
        STA  $6001,Y
COUNT   INY             ; Y+1⇒Y
        CPY  $6000      ; 一轮比完否?
        BNE  LOOP2      ;否,继续比
        INC  $06        ; 是，Y+1，X+1准备比下一轮
        INX
        CPX  $07        ; 最后一轮比完否?
        BNE  LOOP1      ; 否，继续比
        BRK             ; 是，结束
```

截止目前为止，在我们所遇到的程序中,关于数的比较问题，讨论的都是无符号数。当两个无符号数比较时（A-B)应由标志C来判断大小，C=1则A≥B，C=0则A<B。但是当两个有符号数（最高位为符号位，负数用补码表示）比较大小时，就不能用标志位C来判断了，而应由N和V两个标志位联合判断。具体的判断方法是当N，V两标志位相同时（即N=1,V=1或N=0，V=0)

则A≥B，当N，V两标志位不同时（即N=1，V=0或N=0，V=1）则A<B。这可用两数相减时的几种情况加以说明：

（1）参加比较（A－B）的两数，都是正数或者都是负数，这肯定不会产生溢出，所以总有V=0，那么就可根据减操作形成的符号标志N来判断，N=0说明减结果为正，则A≥B；N=1说明减结果为负，则A<B。

（2）参加比较（A－B）的两数中A>0，B<0，这个结果显然应该是A>B，但也有两种具体情况，一种情况是无溢出V=0，例如A=01，B=FE（－02的补码），A－B的结果写成二进制是00000011，标志位N=0，这符合V=0，N=0则A≥B的情况；另一种情况是有溢出V=1，例如A=7F，B=FE（－02的补码）A－B的结果写成二进制是10000001，标志位N=1，这符合V=1，N=1 则A≥B的情况。

（3）参加比较（A－B）的两数中A<0，B>0，这个结果显然应当是A<B，但也有两种具体情况，一种情况是无溢出V=0，例如A=FF（－01的补码），B=02，A－B的结果写成二进制是11111101，标志位N=1，这符合V=0，N=1则A<B的情况；另一种情况是有溢出V=1，例如A=FE（－02的补码），B=7F，A－B的结果写成二进制是01111111，标志位N=0，这符合V=1，N=0则A<B的情况。

在下面的程序举例中，将用N，V标志位联合判断的办法来对有符号数进行比较。

还需说明的一点是由于CMP指令不影响标志位V，所以在两有符号数比较时不能用CMP指令，而要用SBC指令来完成两数相减的操作。

例8 有符号数组找最大值

仍用本节例2中的实际数据，只不过6001～6005单元中5个数看成是有符号数，因此程序执行结果应为（06）=79。

源程序：

```
        ORG  $0300
        LDY  #00
        LDX  $6000
        DEX
        LDA  $6001,Y；设第一个数为最大数
        STA  $06     ；(06)=最大数
LOOP    LDA  $06
        INY
        SEC
        SBC  $6001,Y；取一数和取到A中的最大数比较，
                       哪个大？
        BMI TESTV    ；N=1，再查V
        BVS CHANGE；N=0，V=1，A小，更换 最大数
        JMP COUNT    ；N=0，V=0，A 大，保持最大数
TESTV   BVS COUNT    ；N=1，V=1，A大，保持最大数
CHANGE  LDA  $6001,Y；N=1，V=0或N=0，V=1，A小，
                       更换最大数
        STA  $06
COUNT   DEX          ；所有的数都比较完毕否？
        BNE LOOP     ；否，继续比较
        BRK          ；是，结束
```

例9 8位有符号数排序

仍用本节例7中的实际数据，只不过6001～6006单元中的数看成是有符号数，因此程序执行结果应为：

(6001)=B5

(6002)=D1

(6003)=19

(6004) = 2A

(6005) = 3F

(6006) = 60

源程序：

```
        ORG  $0300
        LDA  $6000
        STA  $07
        DEC  $07          ; (07) = N - 1
        LDA  #01
        STA  $06          ; (06) = 01
        LDX  #00          ; X = 00
LOOP1   LDY  $06
LOOP2   SEC
        LDA  $6001,X      ; 前一数与后面各数分别比较
        SBC  $6001,Y      ; 哪个数大?
        BVS  TESTN        ; V = 0再查N
        BPL  NOCHG        ; V = 1, N = 0前一数小
        BMI  CHG          ; V = 1, N = 1前一数大交换位置
TESTN   BMI  NOCHG        ; V = 0, N = 1前一数小
CHG     LDA  $6001, X     ; 交换位置
        PHA
        LDA  $6001, Y
        STA  $6001, X
        PLA
        STA  $6001, Y
NOCHG   INY               ; Y + 1⇒Y
        CPY  $6000        ; 一轮比完否?
        BNE  LOOP2        ; 否，继续比较
```

```
INC  $06     ；是，Y+1，X+1准备比下一轮
INX
CPX  $07     ；最后一轮比完否？
BNE  LOOP1   ；否，继续比
BRK          ；是，结束
```

§2—4 代码转换程序

代码转换是微型计算机应用中常遇到的问题。外围设备以ASCII，BCD或各种专用代码的形式向计算机提供数据，计算机系统必须将这些数据转换成可供处理的代码形式。输出设备需要ASCII，BCD，七段代码或其它代码的数据，所以计算机就必须在处理结束之后将结果转换成合适的代码形式送给输出设备。此外，在处理数据时，有时需用二进制数，有时需用十进制数，这就要求计算机根据具体需要进行代码转换。

代码转换可以采用包括算术和逻辑函数的算法加以处理，亦可采用查表的方法来进行处理。查表方法只需做简单的程序设计，但表格要占用大量的存贮单元。以下将对这两种方法加以介绍。

例1 二进制整数转换成十进制整数。

目的：将存放在6000单元中的8位二进制整数转换成三位BCD码整数，并将这三位结果按高位至低位的顺序依次存放在6001单元低4位及6002单元中

示范题：a)（6000）=17　　（十六进制）

结果（6001）=00　　（十进制）

（6002）=23　　（十进制）

b)（6000）=FC　　（十六进制）

结果（6001）=02　　（十进制）

（6002）=52　　（十进制）

c) (6000) = 80　　(十六进制)

结果 (6001) = 01　　(十进制)

(6002) = 28　　(十进制)

转换方法：设8位二进制整数为：

$N_{二} = a_7a_6a_5a_4a_3a_2a_1a_0$

其对应的十进制整数应为：

$$N_{十} = a_72^7 + a_62^6 + a_52^5 + a_42^4 + a_32^3 + a_22^2 + a_12^1 + a_02^0 = ((((((a_72 + a_6)2 + a_5)2 + a_4)2 + a_3)2 + a_2)2 + a_1)2 + a_0$$

框图：

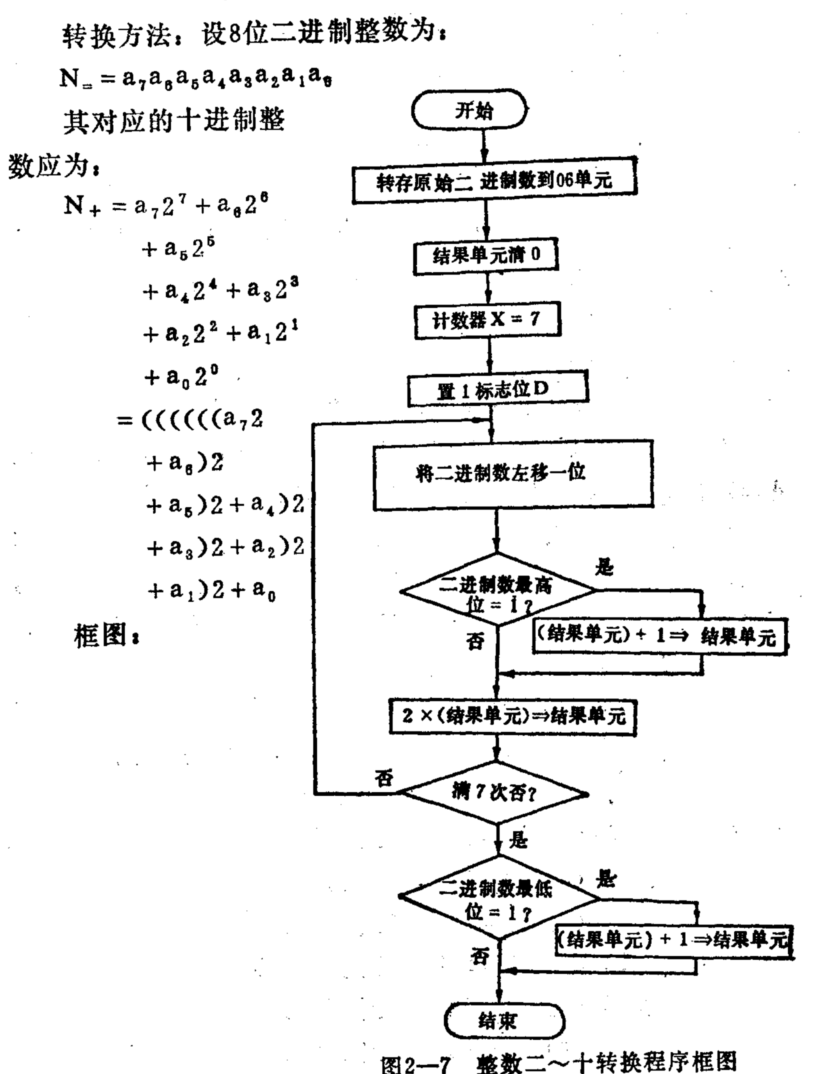


图2—7　整数二～十转换程序框图

源程序:

```
        ORG   $0300
        LDA   $6000     ; 将原始二进制数转存到
        STA   $06         06工作单元
        LDA   #00
        STA   $6001     ; 将结果单元6001清0
        STA   $6002     ; 将结果单元6002清0
        SED             ; 置1十进制运算标志位
        LDX   #07       ; 预置计数器次数为7
LOOP1   ASL   $06       ; 将二进制数左移1位
        BCC   LOOP2     ; 二进制数最高位为0转
                          LOOP2
        CLC             ; 二进制数最高位为1,
        LDA   $6002       把结果单元低位加1后
        ADC   #01         再送回结果单元
        STA   $6002
        LDA   $6001
        ADC   #00
        STA   $6001
LOOP2   LDA   $6002     ; 用自身相加的方法
        ADC   $6002       来实现(结果单元)×2
        STA   $6002
        LDA   $6001
        ADC   $6001
        STA   $6001
        DEX             ; 满7次否?
        BNE   LOOP1     ; 未满, 继续循环
        LDA   $06       ; 满, 判二进制数最低位
```

```
                                        为1否？
        BPL     DONE                  ；为0，转结束
                                      ；为1，将结果单元内容
                                        +1
        LDA     $6002
        ADC     #01
        STA     $6002
        LDA     $6001                 ；加进位
        ADC     #00
        STA     $6001
DONE    BRK
```

应当注意到在这个程序中的加法用的是十进制加法，(结果单元)×2这一步并不采用左移一位的办法，而采用十进制的自身相加步骤来实现，这是二→十进制转换所需要的。此外在程序开始处转存二进制数到06单元去是为了防止执行程序后丢失原始数据。

对于8位二进制整数～三位十进制整数的转换，我们还可以采用查表的方法来实现。说明如下：

设8位二进制整数存放在BIN单元中，转换后的结果三位十进制整数存放在BCD1单元的低四位及BCD2单元中。

把8位二进制整数各位的权，按由高位至低位的顺序，用BCD码依次放在START～START+8的9个单元中，构成一个权表(最高位的权为128占用两个单元)。即

START	01
START + 1	28
START + 2	64
START + 3	32
START + 4	16
START + 5	08
START + 6	04
START + 7	02
START + 8	01

权表中的数为BCD码。

根据：

$$(a_7a_6a_5a_4a_3a_2a_1a_0) = = (a_7 2^7 + a_6 2^6 + a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0)_{+}$$

可得框图及源程序如下：

框图：

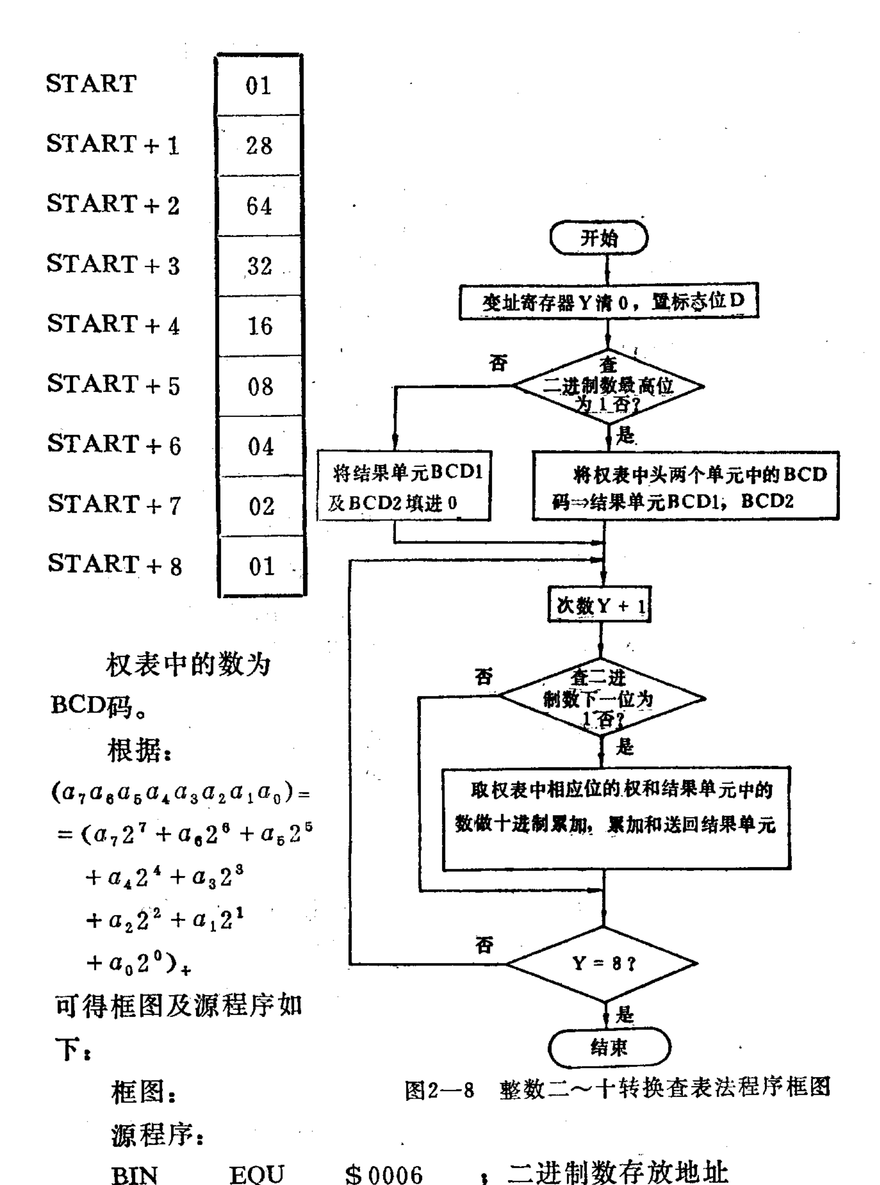


图2—8　整数二～十转换查表法程序框图

源程序：

```
BIN     EQU     $0006     ；二进制数存放地址
BCD1    EQU     $0007     ；十进制数百位存放地址
```

```
BCD2    EQU   $0008      ; 十进制数十位、个位存放地
                         址
TEMP    EQU   $0009      ; 工作单元地址
        ORG   $0300      ; 程序启动地址
        LDA   BIN        ; 取待转换二进制数，
        STA   TEMP       送到中间工作单元
        LDY   #00        ; 清计数器
        SED              ; 置1D标志位，准备做十
                         进制加法
        ASL   TEMP       ; 左移二进制数， 判二进制
                         数最高位为1否
        BCC   LOOP1      ; 为0转LOOP1
        LDA   START      ; 为1，取权表中最高位权
                         的高两位BCD码
        STA   BCD1       ; 送到结果单元BCD1
        INY
        LDA   START, Y   ;取权表中最高位权的低两位
                         BCD码
        STA   BCD2       ; 送到结果单元BCD2
        JMP   LOOP2
LOOP1   LDA   #00        ; 二进制数最高位为0
        STA   BCD2       ; 使结果单元BCD2为0
        STA   BCD1       ; 使结果单元BCD1为0
        INY
LOOP2   INY
        ASL   TEMP       ; 查二进制数下一位为1否
        BCC   LOOP3      ; 为0转LOOP3
        CLC
```

```
        LDA   START,Y; 为1，取权表中下一位的权和结
        ADC   BCD2     ; 果单元中的数累加
        STA   BCD2
        BCC   LOOP3
        INC   BCD1
LOOP3   CPY   #08      ; 计数够8次否?
        BNE   LOOP2    ; 不够，继续查表
        BRK            ; 够，结束
START   DFB   $01, $28, $64, $32, $16
        DFB   $08, $04, $02, $01;权表(表中各数为
                                   BCD码)
```

注意，此程序中的加法用十进制加法。

例2 二进制小数转换成十进制小数

目的：把存放在BIN单元中的8位二进制小数转换成三位BCD码小数，并将这三位BCD码小数按高位至低位的顺序存放到BCD，BCD+1，BCD+2三个单元中。

示范题：*a*) (BIN)=FF (十六进制)

结果 (BCD)=09 (十进制)

(BCD+1)=09 (十进制)

(BCD+2)=06 (十进制)

即 $(0.11111111)_2=(0.996)_{10}$

b) (BIN)=A0 (十六进制)

结果 (BCD)=06 (十进制)

(BCD+1)=02 (十进制)

(BCD+2)=05 (十进制)

即 $(0.10100000)_2=(0.625)_{10}$

转换方法：设8位二进制小数为X，转换后的三位十进制小数为Y，则有：

$$X = 0.a_1a_2a_3a_4a_5a_6a_7a_8$$
$$= a_1 \times 2^{-1} + a_2 \times 2^{-2} + a_3 \times 2^{-3} + a_4 \times 2^{-4} + a_5 \times 2^{-5} + a_6 \times 2^{-6} + a_7 \times 2^{-7} + a_8 \times 2^{-8}$$

（其中a_1～a_8为0或1的二进制数）

$$Y = 0.b_1b_2b_3$$
$$= b_1 \times 10^{-1} + b_2 \times 10^{-2} + b_3 \times 10^{-3}$$

（其中b_1～b_3为0～9的十进制数）

应有Y = X，即：

$$b_1 10^{-1} + b_2 10^{-2} + b_3 10^{-3} = a_1 2^{-1} + a_2 2^{-2} + a_3 2^{-3} + a_4 2^{-4} + a_5 2^{-5} + a_6 2^{-6} + a_7 2^{-7} + a_8 2^{-8}$$

等式左边，右边均乘以$(10)_{十} = (1010)_{二}$等式仍应相等，即：

$$(10)_{十}(b_1 10^{-1} + b_2 10^{-2} + b_3 10^{-3}) = (1010)_{二}(a_1 2^{-1} + a_2 2^{-2} + \cdots + a_8 2^{-8})$$

$$b_1 + (b_2 10^{-1} + b_3 10^{-2}) = (1000 + 0010)_{二}(a_1 2^{-1} + a_2 2^{-2} + \cdots + a_8 2^{-8})$$
$$= (2^3 + 2^1)(a_1 2^{-1} + a_2 2^{-2} + + \cdots + a_8 2^{-8})$$
$$= a_1 2^2 + a_2 2^1 + a_3 2^0 + a_4 2^{-1} + a_5 2^{-2} + a_6 2^{-3} + a_7 2^{-4} + a_8 2^{-5} + a_1 2^0 + a_2 2^{-1} + a_3 2^{-2} + a_4 2^{-3} + a_5 2^{-4} + a_6 2^{-5} + a_7 2^{-6} + a_8 2^{-7}$$

整理以后得：

$$b_1 + (b_2 10^{-1} + b_3 10^{-2}) = a_1 2^2 + a_2 2^1 + (a_3 + a_1) 2^0 + + (a_4 + a_2) 2^{-1} + (a_5 + a_3) 2^{-2} + (a_6 + a_4) 2^{-3} + (a_7 + a_5) 2^{-4} + (a_8 + a_6) 2^{-5} + a_7 2^{-6} + a_8 2^{-7}$$

等式两边大于及等于1的数应该相等，所以：

$$b_1 = a_1 2^2 + a_2 2^1 + (a_3 + a_1)2^0 + a_c 2^{-1}$$

a_c是2^{-1}项相加后的进位项($a_4 + a_2$有进位时$a_c 2^{-1} = 1$)以上的运算表明，将二进制小数$0.a_1a_2a_3a_4a_5a_6a_7a_8$转换为十进制小数只需将二进制小数乘以$(1010)_二$其大于及等于1的整数部分就是十进制小数的第一位$b_1$。

上列等式左右两边除了大于1的部分相等以外，小于1的部分也应相等，即：

$$\begin{aligned} & b_2 10^{-1} + b_3 10^{-2} \\ = & (a_4 + a_2)2^{-1} + (a_5 + a_3)2^{-2} \\ & + (a_6 + a_4)2^{-3} + (a_7 + a_5)2^{-4} \\ & + (a_8 + a_6)2^{-5} + a_7 2^{-6} \\ & + a_8 2^{-7} \end{aligned}$$

如果将这个式子再度在等式两边乘以$(10)_十 = (1010)_二$，则等式右边大于及等于1的部分就是十进制小数的第二位b_2。如果再将剩下的小于1的部分再次乘$(10)_十 = (1010)_二$，那么大于等于1的部分就是十进制小数的第三位b_3。可知，只需做三次乘拾取整就可将八位二进制小数转换成为三位十进制小数。

框图：

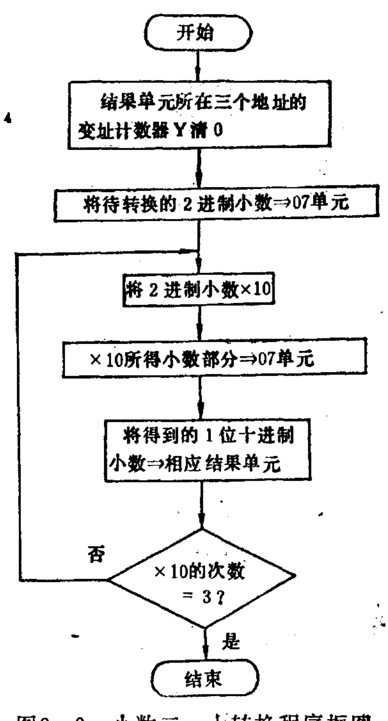


图2—9　小数二～十转换程序框图

源程序：

```
BIN     EQU   $6000    ；二进制小数所在单元地址
BCD     EQU   $6001    ；十进制小数所在单元首地址
        ORG   $0300    ；程序启动地址
        LDY   #00      ；结果单元变址计数预置
```

```
        LDA   BIN        ; 将二进制小数取到
        STA   $07          工作单元07去
NEXT    JSR   MU10       ; 转×10子程序
        STA   BCD, Y     ; 将×10后所得整数⇒结果单元
        LDA   $07        ; 将上次×10后所得小数部分
                           ⇒A
        INY
        CPY   #03        ; ×10满三次了吗?
        BNE   NEXT       ; 不满，再继续×10
        BRK              ; 满，程序结束
MU10    LDX   #00        ; 采用二进制小数自身加10次
                           的办法实现×10
        STX   $06        ; 存放每次×10后所得整数的
                           06单元清0
        LDX   #09        ; 相加次数预置
        CLD
LOOP1   CLC
        ADC   $07        ; 对二进制小数作加法
        BCC   LOOP2      ; 相加后无进位转LOOP2
        INC   $06        ; 相加后有进位则将(06单元)
                           +1，这就是取整数部分
LOOP2   DEX              ; 二进制小数本身相加够十次
                           了吗
        BNE   LOOP1      ; 不够再继续做加法
        STA   $07        ; 够将×10后所得小数送07单元
        LDA   $06        ; 将×10后所得整数从06单元取
                           到A
        RTS              ; 返回
```

.END

注意此程序中的加法是用的二进制加法。

例3　三位十进制整数转换成二进制整数(十进制数≤255)

目的：将由高位至低位依次存放在存贮单元ADR，ADR+1，ADR+2中的三位不大于255 的 BCD 整数转换成二进制整数并将转换结果存放在ADR+3单元。

示范题：　(ADR)=02　　(十进制)

　　　　(ADR+1)=05　　(十进制)

　　　　(ADR+2)=05　　(十进制)

结果：　(ADR+3)=FF　　(十六进制)

转换方法：采用乘拾加数的方法来进行整数十～二转换，以十进制数 359 如何用此法转换成二进制数 101100111 为例加以说明。

$$(359)_{十}=3\times100+5\times10+9=(3\times10+5)\times10+9$$

可见359是由两次乘拾加数的运算而得其值，其中第一次乘拾加数是$A=3\times10+5$

第二次乘拾加数是$B=(A\times10)+9$ (此即最终结果值)，我们若把上述算式中的十进制数3，5，9都用BCD码表示成 0011，0101，1001，算式中的十进制数10表示成1010，并按二进制数运算规则进行乘拾加数的运算，那么运算的结果就是十进制数 359 所对应的二进制数了。仍以359为例

$$\begin{aligned}(359)_{十}&=(3\times10+5)\times10+9\\&=(0011\times1010+0101)\times1010+1001\\&=(11110+0101)\times1010+1001\\&=100011\times1010+1001\\&=101011110+1001=(101100111)_{二}\end{aligned}$$

所以三位十进制整数 359 转换成二进制整数是九位。一个存贮单元容纳不了，本例中存放转换结果的存贮单元只给了一个

ADR3单元，故对于待转换的十进制数要限制在255以内。

框图：

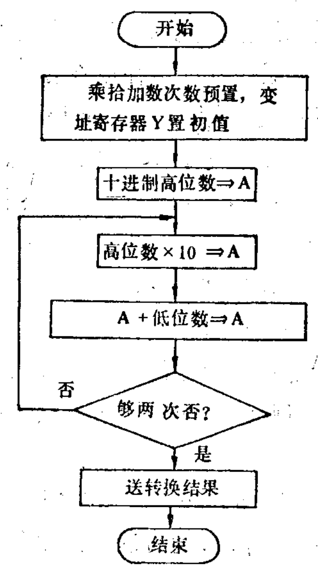


图2—10 整数二～十转换程序框图

源程序：

```
ADR    EQU   $6000
       ORG   $0300
       CLD
       LDX   #02       ；乘拾加数次数预置
       LDY   #00       ；变址计数器预置
       LDA   ADR，Y    ；取十进制百位数
LOOP   ASL   A         ；得2A
       STA   $06       ；将2A送工作单元06
       ASL   A
       ASL   A         ；得8A
       CLC
```

```
ADC    $06       ; 得2A+8A=10A完成乘拾
INY
ADC    ADR, Y    ; 把乘拾结果再加数
DEX              ; 乘拾加数次数够2次了吗?
BNE    LOOP      ; 不够，再继续做
INY              ; 够，准备好存结果地址
STA    ADR, Y    ; 存转换结果
BRK
```

以上是用于≤255的三位十进制数转换成8位二进制数的程序。而三位十进制数可以到达999的范围，所以将大于255的三位十进制数转换成二进制数时必须为二进制数安排两个地址单元，这种情况将在§2—5例3中再作讨论。

例4 十进制小数转换成二进制小数

目的：将按高位至低位次序顺序放在BCDH单元中低四位和BCDL单元中的共计三位BCD码小数转换成八位二进制小数并存放在RESU单元中

示范题：(BCDH) = 07 (十进制)

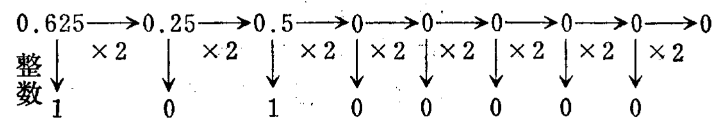
(BCDL) = 50 (十进制)

结果 (RESU) = C0 (十六进制)

即 $(0.750)_{十} = (0.11000000)_{二}$

转换方法：采用将十进制小数乘 2 取整的方法来实现转换，即把待转换的十进制小数采用连续乘以 2 而记录其乘积中整数的方法，以十进制小数0.625为例

小数部分



所以十进制小数0.625经过八次乘2取整转换成8位二进制小数

0.10100000

框图:

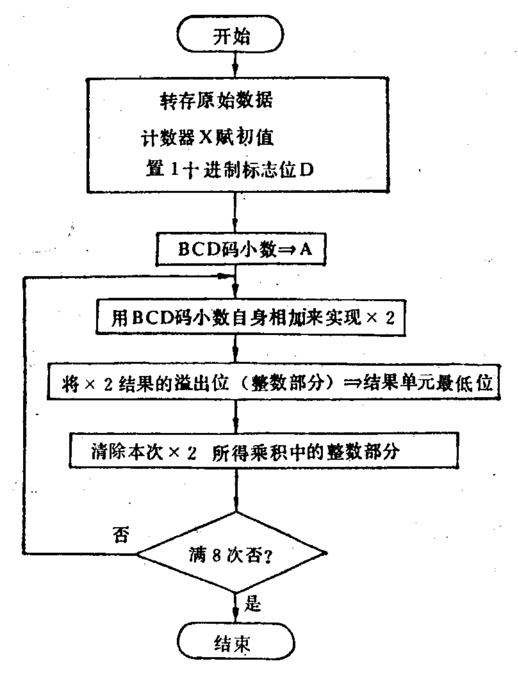


图2—11 小数十～二转换程序框图

源程序:

```
BCDH   EQU   $6000   ; BCD码小数最高位地址
BCDL   EQU   $6001   ; BCD码小数低两位地址
RESU   EQU   $6002   ; 二进制小数地址
BCD1   EQU   $06     ; 工作单元
BCD2   EQU   $07     ; 工作单元
       ORG   $0300   ; 程序启动地址
       LDA   BCDH    ; 将原始数据转存
```

```
        STA    BCD1          以防原始数据丢失
        LDA    BCDL
        STA    BCD2
        LDX    #08      ；计数器赋初值
        SED             ；置1标志位D
NEXT    CLC
        LDA    BCD2     ；用十进制加法实现乘2
        ADC    BCD2
        STA    BCD2
        LDA    BCD1
        ADC    BCD1
        STA    BCD1
        LDA    #$10     ；判×2后所得整数部分是1否
        BIT    BCD1
        BEQ    LOOP1    ；是0转LOOP1去使C=0
        SEC             ；是1，使C=1
        JMP    LOOP2    ；转LOOP2
LOOP1   CLC
LOOP2   ROL    RESU     ；把×2所得整数⟹结果单元
                          最低位
        LDA    #$0F     ；把×2乘积单元中的整数部分
                          清除
        AND    BCD1     ；使只剩下小数部分
        STA    BCD1
        DEX             ；够8次否？
        BNE    NEXT     ；不够，继续循环
        BRK             ；够，结束
        .END
```

注意此程序中的×2是用的十进制小数自身作十进制加法来实现的。

例5　十进制转换成七段代码

目的：将存贮单元6000中的一个十进制数（高4位为0）转换成七段代码，结果存入存贮单元6001。若存贮单元6000中包含的不是一个十进制数字，则存贮单元6001清0

转换方法：七段代码是七段发光二极管显示器显示字符时所采用的代码，对共阴极七段显示器所用七段代码显示十进制数字的具体安排见图2—12

数　字	代　码
0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F

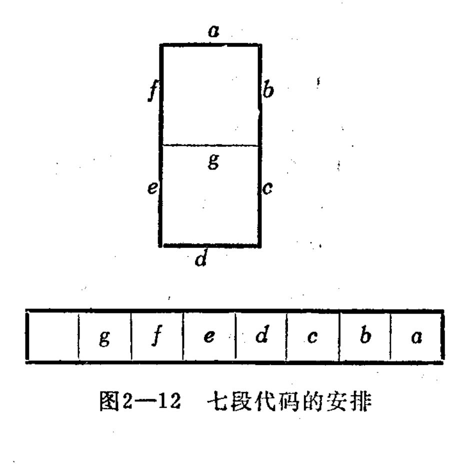


图2—12　七段代码的安排

如图所示，在一个字节中构成七段代码的方法是：最高位总是0，其它七位由低位向高位顺序为*a*、*b*、*c*、*d*、*e*、*f*、*g*各段的代码（为1亮，为0不亮）。注意表中用7D而不是用7C（顶段不亮）代表6，以免与小写的*b*混淆；用6F而不是用67（底段不亮）代表9，这点没有特殊的理由。在编制程序时，需在内存中安排一张这样的七段码表。

示范题：　　a)　　(6000) = 03

　　　　　　结果　(6001) = 4F

b)　　(6000) = 28

结果　(6001) = 00

框图：

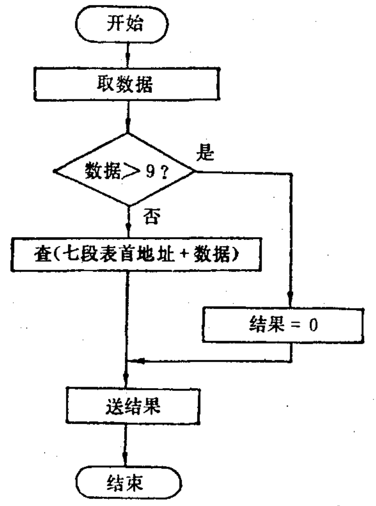


图2—13　十进制～七段码转换程序框图

源程序：

```
        ORG   $0300
        LDA   #00
        LDX   $6000
        CPX   #10
        BCS   DONE          ; C=1不是十进制数字
        LDA   SSEG, X
DONE    STA   $6001
        BRK
SSEG    DFB   $3F, $06, $5B, $4F, $66  ;七段码表
        DFB   $6D, $7D, $07, $7F, $6F
```

欲把此程序改为十六进制～七段码的转换程序，只要把七段

表中再加上A～F这六个数字的对应七段码，并把程序略作变动即可。

例6　十六进制转换成ASCII码

目的：将存贮单元6000的内容转换成 ASCII码字符。存贮单元6000中有一个16进制数字（高4位为0），将此数字对应的 ASCII字符存入存贮单元6001中。

示范题：a)　　(6000) = 06

　　　　结果　(6001) = 36 = ‘6’

表2—1　　　　ASCII 字 符 编 码 表

ASCII CHARACTER SET(7—BIT CODE)

MSD / LSD		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P		*p*
1	0001	SOH	DC1	!	1	A	Q	*a*	*q*
2	0010	STX	DC2	″	2	B	R	*b*	*r*
3	0011	ETX	DC3	#	3	C	S	*c*	*s*
4	0100	EOT	DC4	$	4	D	T	*d*	*t*
5	0101	ENQ	NAK	%	5	E	U	*e*	*u*
6	0110	ACK	SYN	&	6	F	V	*f*	*v*
7	0111	BEL	ETB	′	7	G	W	*g*	*w*
8	1000	BS	CAN	(	8	H	X	*h*	*x*
9	1001	HT	EM	)	9	I	Y	*i*	*y*
A	1010	LF	SUB	*	:	J	Z	*j*	*z*
B	1011	VT	ESC	+	;	K	[	*k*	{
C	1100	FF	FS	,	<	L	＼	*l*	|
D	1101	CR	GS	−	=	M	]	*m*	}
E	1110	SO	RS	。	>	N	↑	*n*	～
F	1111	SI	US	/	?	O	←	*o*	DEL

b)　　(6000) = 0C

结果　(6001) = 43 = ‘C’

转换方法：编制此程序的基本思想是把数字0的ASCII码加到所有待转换的十六进制数字上，以使十六进制数字转换成所对应的ASCII码。但是这一加法仅使十六进制数字中的0～9得到了正确转换，而对于A～F之间的数的转换不能得到正确结果。阅读7单位码ASCII字符编码表（表2—1）就可看到在9～A之间（对应的ASCII码为39～41之间）有一个间隔。编制程序时必须把这个间隔量加上去。

框图：

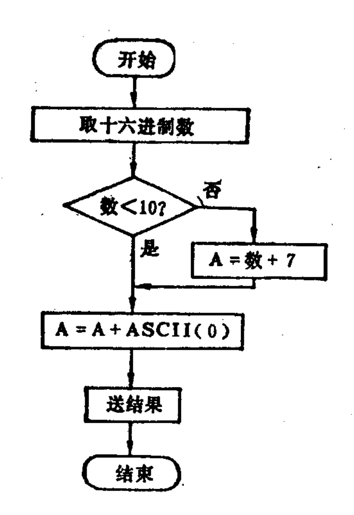


图2—14　十六进制～ASCII码程序框图

源程序：

```
ORG   $0300
LDA   $6000    ；取出待转换的十六进制数
CMP   #$0A     ；数＜A吗？
BCC   ASCZ     ；是，转ASCZ
```

```
        CLC                  ；否，加一个间隔量
        ADC    #07
ASCZ    ADC    #'0           ；加数字0对应的ASCII码
        STA    $6001         ；存结果
        BRK
```

需要说明一下，ASCII编码表只规定bit6～bit0这七位编码，最高位 bit7在APPLE Ⅱ 中被规定为1，而在其它计算机中往往规定为0。上面这个程序在APPLE II中运行后，得到的是APPLE Ⅱ的ASCII码，（因为'0＝B0）。

例7 十进制数转换成ASCII码

目的：若要求把100个存贮单元中用BCD码表示的十进制数转换成 ASCII码表示，每一单元内是两个十进制数，起始地址是$6000，转换后的ASCII码存放到起始地址为＄7000的200个存贮单元中去(低半字节转换在前)。

源程序：

```
SOURCE  EQU   $6000        ；源地址
DESTI   EQU   $7000        ；目标地址
        ORG   $0300
        LDX   #00          ；源地址计数初值
        LDY   #00          ；目标地址计数初值
NEXT    LDA   SOURCE, X    ；取十进制数
        AND   #$0F         ；屏蔽高四位
        ORA   #$B0         ；低位数的ASCII码
        STA   DESTI, Y     ；存结果
        INY
        LDA   SOURCE, X    ；再取原十进制数
        LSR   A            ；将高四位移至低四位
        LSR   A
```

```
        LSR   A
        LSR   A
        ORA   #$B0        ; 高位数的ASCII码
        STA   DESTI, Y    ; 存结果
        INY
        INX
        CPX   #$64
        BNE   NEXT        ; 转换次数不够，继续转换
        BRK
```

§2—5 子程序

在程序编制过程中，常常会遇到相同的计算或操作。例如代码转换、作乘法、除法、求三角函数等。可以把这些相同的部分编制成一个独立的程序段称为子程序，整个程序由主程序和子程序构成。当遇到相同的计算或操作时就转入子程序，而不必对这些相同部分重复编制程序了。大多数程序都是由一个主程序和几个子程序组成，使用子程序的程序称为主程序。在主程序中如果要用到子程序就把控制转移到子程序，这称为转子。当子程序执行完了之后再返回到主程序，这称为返主。

6502微处理机备有专门的转子指令JSR，用以把控制转移到子程序，并备有专门的子程序返回指令RTS，用以使子程序返回主程序。JSR指令把子程序起始地址放入程序计数器PC之前先把程序计数器原来的值(返主地址)存入堆栈，RTS指令则从堆栈中取出返主地址并把它放到程序计数器PC中。(关于JSR和RTS两条指令的分析请见第一章§1—2)。子程序的结构应当如图2—15所示：在称为子程序入口处的第一条指令的地方应给予一个标号，而在这段子程序的结束处则应放置一条返回指令RTS，当主程序要调

用子程序时，应该事先由程序员或是由计算机的监控程序预置堆栈，这样才能保证返回地址保留在合适的位置。

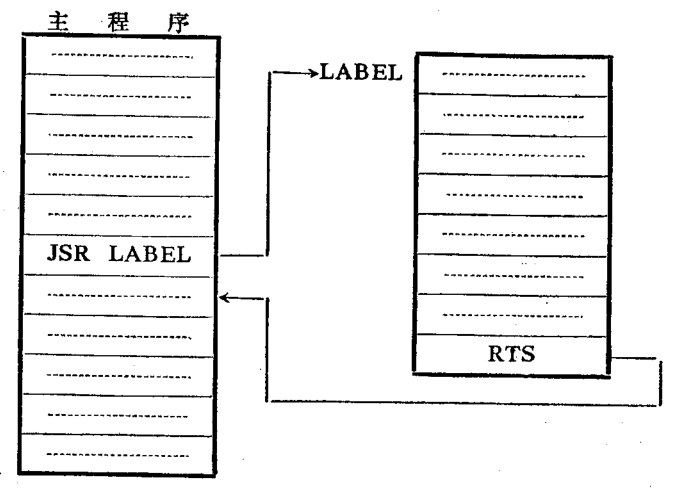


图2—15 主程序与子程序关系图

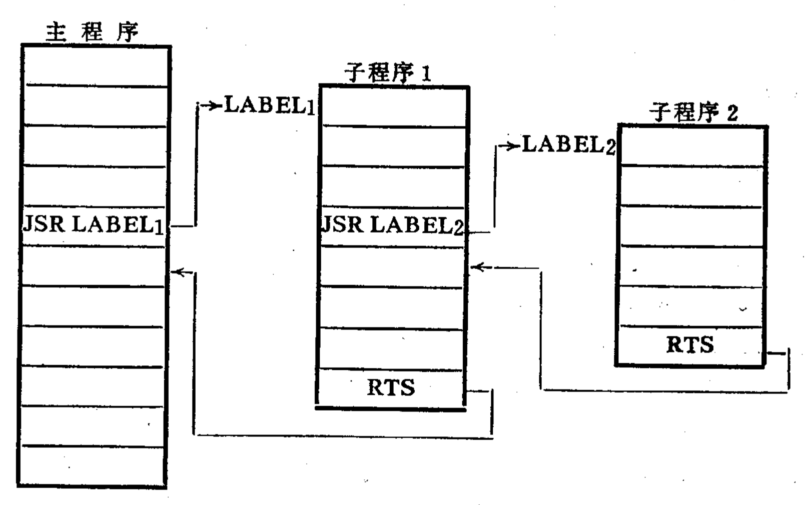


图2—16 子程序的嵌套

在实际问题中，常常会出现这样的情况：主程序工作时，需要转入子程序1，而子程序1工作时又需要转入子程序2。当子程序2工作完毕后返回子程序1，子程序1执行完毕后返回主程序。在这个过程中，子程序1相对于子程序2而言就处在主程序的地位上了。这叫做子程序的嵌套。图2—16表示的是两重子程序嵌套的情况，仿此，还可以构成多重子程序嵌套的情况。

关于主程序和子程序的衔接问题，还应注意的是主程序送给子程序计算或操作的数据放在何处，子程序计算或操作后的结果放在何处。对于这两个问题，主程序和子程序必须按照约定，相互之间衔接好，程序员在编制程序时应予注意。

下面举例说明子程序的编制方法。

例1　找最大值

目的：找一个无符号数据块中的最大值，数据块的长度放在6000存贮单元，数据块起始地址为6001。最大值存于06单元。

示范题：(6000) = 05　　(数据块长度)

(6001) = 67

(6002) = 79

(6003) = 15

(6004) = E3

(6005) = 72

结果：　(06) = E3　　因为这五个数中的最大数是E3

框图：

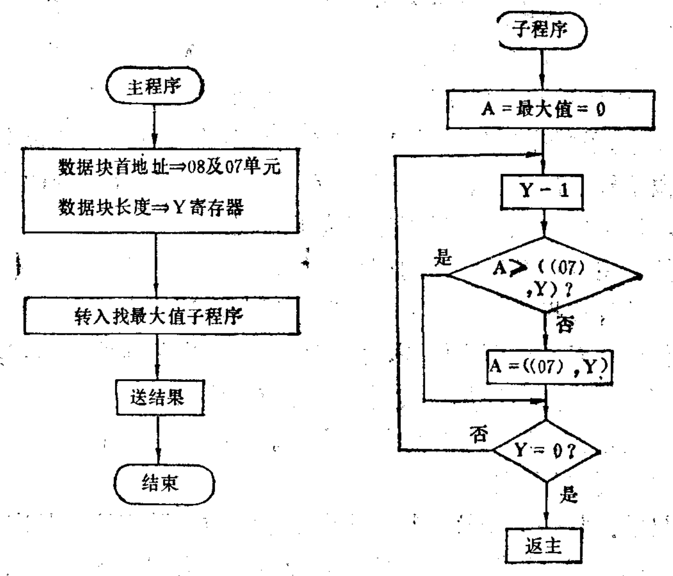


图2—17 找最大值程序框图

源程序：

```
      ORG   $0300
      LDA   #01     ; 把数据块首地址送入08和07单元
      STA   $07     ; 即(08)(07)=6001
      LDA   #$60
      STA   $08
      LDY   $6000   ; 把数据块长度存于Y寄存器
      JSR   MAXM    ; 转子程序
      STA   $06     ; 存结果
      BRK
MAXM  LDA   #00     ; 假设最大值为0放在A中
CMPE  DEY           ; Y-1
      PHP           ; 标志寄存器进栈，保护Z标志
```

```
        CMP   ($07),Y ；下一个数>最大值吗？
        BCS   NOCHG   ；不是，转NOCHG保留最大值不变
        LDA   ($07),Y ；是，换最大值放入A中
NOCHG   PLP           ；标志寄存器出栈，判上面Y-1结果
                        是否为0
        BNE   CMPE    ；不为0继续比较
        RTS           ；为0，返回主程序
```

阅读该程序应当注意到主程序在转子前把数据块首地址放到08和07单元中，把数据块长度放在Y中，而子程序把所得最大值结果放在A中，这是为了主程序、子程序的衔接所做的规定，有了这种规定，对于另一个起始地址，另一个长度的数据块，找最大值则只需仿照此例先把起始地址送入08和07单元，长度送入Y，然后转入子程序MAXM即可找到最大值并存放在A中。这就使子程序的使用有了通用性。

例2 比较两个字符串是否相同

目的：比较两个ASCII字符串，看它们是否相同。字符串长度在存贮单元6000中，两个字符串的起始地址各为6100和6200，如果这两个字符串相同，将6001单元清0，否则将6001单元置为FF

示范题：a)

```
        (6000)=03  . 字符串长度
        (6100)=43     'C'
        (6101)=41     'A'
        (6102)=54     'T'
        (6200)=43     'C'
        (6201)=41     'A'
        (6202)=54     'T'
结果：  (6001)=00
```

b)

(6000) = 03

(6100) = 52　　　'R'

(6101) = 41　　　'A'

(6102) = 54　　　'T'

(6200) = 43　　　'C'

(6201) = 41　　　'A'

(6202) = 54　　　'T'

结果：　(6001) = FF

框图：

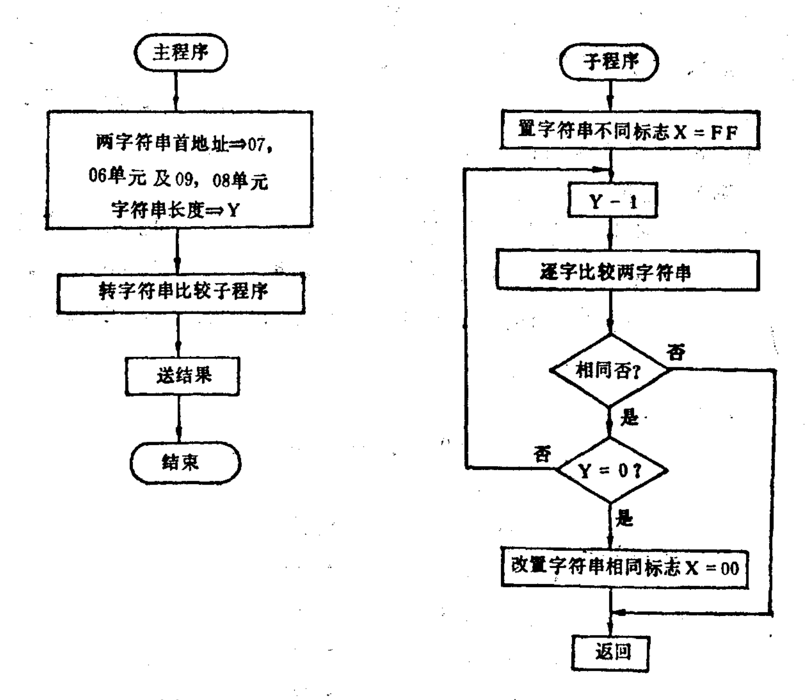


图2—18　比较两个字符串是否相同程序框图

源程序：

```
        ORG   $06
        DW    $6100, $6200; 存两个字符串起始地址到
                          07、06、09、08单元
        ORG   $0300
        LDY   $6000       ; 将字符串长度存入Y寄存
                          器
        JSR   PMTCH       ; 转子程序，检查两个字符
                          串是否相同
        STA   $6001       ; 存结果
        BRK
        ORG   $0350
PMTCH   LDX   #$FF        ; 两字符串不相同的标志存
                          于X
CMPE    DEY               ; Y-1
        LDA   ($06), Y    ; 取一个字符串中一字符
        CMP   ($08), Y    ; 和另一字符串中一字符相
                          比较
        BNE   DONE        ; 不同，转DONE
        TYA               ; 相同，把Y⇒A
        BNE   CMPE        ; Y≠0，继续比较
        LDX   #00         ; Y=0，改送两字符串相同
                          标志到X
DONE    TXA               ; X⇒A
        RTS               ; 返回主程序
```

由此程序可见，在两个字符串比较时，是从后向前比的。

例3　三位十进制整数转换成二进制整数

目的：将由高位至低位依次存放在存贮单元BCDH，BCDM，

BCDL中的三位BCD码整数转换成二进制整数，并将转换结果存于RESH和RESL单元

此程序已在§2—4例3中介绍过，我们此处换成调用子程序的办法来重新编排源程序

源程序：

```
BCDH    EQU   $6000      ; BCD码百位数所在地址
BCDM    EQU   BCDH+1     ; BCD码拾位数所在地址
BCDL    EQU   BCDM+1     ; BCD码个位数所在地址
RESH    EQU   BCDL+1     ; 二进制数高字节所在地址
RESL    EQU   RESH+1     ; 二进制数低字节所在地址
TEMP    EQU   $06        ; 工作单元
        ORG   $0300      ; 主程序起始地址
        LDA   #00        ; 将结果单元和工作单元清0
        STA   RESL
        STA   RESH
        STA   TEMP
        LDA   BCDH       ; 取百位数
        JSR   MU10       ; 转乘10子程序，将百位数×10
        LDA   BCDM       ; 取拾位数
        JSR   ADD        ; 转加数子程序，将×10结果+拾
                         位数
        LDA   RESL
        JSR   MU10       ; 再做一次×10十数
        LDA   BCDL
        JSR   ADD
        BRK
        ORG   $0350      ; 乘10子程序起始地址
MU10    LDX   #09        ; 计数器预置
```

```
        STA    $07        ; 欲×10的数⇒07单元
        CLD
LOOP    CLC               ; 用自身相加十次的方法实现×10
        ADC    $07
        BCC    NOCY1
        INC    TEMP
NOCY1   DEX
        BNE    LOOP
        STA    RESL       ; ×10结果⇒结果单元低字节
        LDA    TEMP       ; ×10结果的进位⇒结果单元高字
                            节
        STA    RESH
        RTS
        ORG    $0380      ; 加数子程序起始地址
        CLD
ADD     CLC
        ADC    RESL       ; A=A+结果单元低字节
        BCC    NOCY2
        INC    RESH       ; 相加结果高字节仍在RESH
NOCY2   STA    RESL       ; 相加结果低字节仍在RESL
        RTS
```

APPLE Ⅱ 为了方便用户，还把其监控程序（APPLE Ⅱ SYSTEM MONITOR）中若干子程序列出清单（表2—2）以供用户调用。

由于子程序清单中不仅列出了子程序入口地址、名称、操作功能，而且列出了子程序所需初始数据应预置在那个寄存器以及子程序执行过程中使用和变更了哪些寄存器，所以用户不需了解这些子程序的内部结构就可很方便的调用这些子程序。

下面列出APPLE Ⅱ 子程序清单，并举例说明调用的方法。

表2—2　　　　APPLEII　子程序清单

入口地址	名　称	功　　能	调用前需预置的寄存器	被更改的寄存器
$FDED	COUT	输出一个字符到被选择的输出设备	字符ASCII码在A中	—
$FDF0	COUT1	输出一个字符到显示器	字符ASCII码在A中	—
$FC62	CR	送一个回车换行符到显示器	—	A，X，Y
$FD8E	CROUT	送一个回车符到被选择的输出设备	—	A＝$8D(CR)
$F940	PRNTYX	将X、Y内容以四位十六进制方式按YYXX格式送到被选择的输出设备	Y＝高字节 X＝低字节	A
$F941	PRNTAX	将A、X内容以四位十六进制方式按AAXX格式送到被选择的输出设备	A＝高字节 X＝低字节	A
$F944	PRNTX	将X内容以两位十六进制方式送到被选择的输出设备	X	A
$FDDA	PRBYTE	将A内容以两位十六进制方式送到被选择的输出设备	A	A
$FDE3	PRHEX	将A的低半字节以一位十六进制方式送到被选择的输出设备	A	A
$F948	PRBLNK	送三个空格符到被选择的输出设备	—	A＝$A0(SP) X＝0
$F94A	PRBL2	输出1～256个空格符到被选择的输出设备	X＝空格数目(X＝0表示256个空格)	A＝$A0(SP) X＝0

续表

入口地址	名　称	功　能	调用前需预置的寄存器	被更改的寄存器
$FF3A	BELL	输出一个响铃字符到被选择的输出设备	——	A = $87(BEL)
$FBDD	BELL1	APPLE Ⅱ 的喇叭持续响1/10秒	——	A、Y
$FD0C	RDKEY	由被选择的输入设备输入一个字符	——	A = 输入字符 Y
$FD1B	KEYIN	读APPLEⅡ键盘	——	A = 输入字符
$FD6A	GETLN	由被选择的输入设备输入一行字符，最多可达 256 个字符	$33单元中放入提示符	A、Y X = 字符个数 $200单元开始为输入字符存放区
$FD67	GETLNZ	先送一个回车符到输出设备，然后转入GETLN	$33单元中放入提示符	同GETLN
$FD6F	GETLN1	不显示提示符，其它同GETLN	——	同GETLN
$FCA8	WAIT	延时，其延时时间为：$(13+13.5A+2.5A^2)/1.023\mu s$	A = 延时值	A = 0
$FF4A	SAVE	将寄存器内容存在0页以下单元中： A：$45 X：$46 Y：$47 P：$48 S：$49	——	A、X
$FF3F	RESTORE	把执行SAVE子程序存到$45~$49单元中各寄存器的内容送回到对应的各寄存器中	——	A,X,Y,P,S

注：表中所列被选择输入设备一般为键盘，被选择输出设备一般为显示器，在BELL中为喇叭。

下面举例说明调用APPLE Ⅱ 子程序的方法

例4 APPLE Ⅱ 输入码转换程序(一)

目的：由读键盘子程序取在键盘按下的一位十进制数（读入的键码是ASCII码）并进行ASCII～二进制代码转换，转换结果送到A中。

示范题：

a）按键8

结果：A＝08

b）按键5

结果：A＝05

框图：

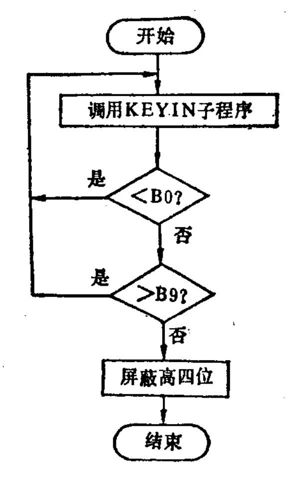


图2—19 APPLE Ⅱ 输入码转换程序(一)框图

源程序：

```
        ORG   $0300
NEWDIG  JSR   $FD1B     ；调用KEYIN 读键盘输入的一
                          个数字到A
```

```
        CMP    #$B0       ; 小于B0码?
        BCC    NEWDIG     ; 是，返回重输入一个新数
        CMP    #$BA       ; 大于B9吗?
        BCS    NEWDIG     ; 是，返回重输入一个新数
        AND    #$0F       ; 是在0～9范围的数,去掉高四
        BRK                 位
```

例5 APPLE Ⅱ 输入码转换程序(二)

目的：取在键盘按下的三位十进制数，并将它们的ASCII码转换为8位二进制数放在A中。

示范题：

a) 按键 255 (十进制)

结果 A＝FF (十六进制)

b) 按键 128 (十进制)

结果 A＝80 (十六进制)

框图：

其中子程序 NEWDIG 框图同例4，子程序MULT10框图略。

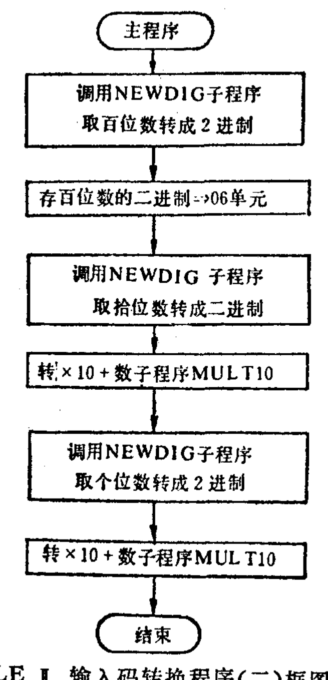


图2—20 APPLE Ⅱ 输入码转换程序(二)框图

源程序：

```
        ORG    $0300
        JSR    NEWDIG     ; 转子，取百位数转成二进制
                            在A
        STA    $06        ; 存入06单元作为中间结果
```

```
        JSR   NEWDIG     ；转子，取拾位数转成二进制
                           在A
        JSR   MULT10     ；转×10+数子程序,得到的中
                           间结果存在06单元
        JSR   NEWDIG     ；转子,取个位数转成二进制数
                           在A
        JSR   MULT10     ；转×10+数子程序得到最后
                           结果在A
        BRK
NEWDIG  JSR   $FD1B      ；NEWDIG子程序作用见例3
        CMP   #$B0
        BCC   NEWDIG
        CMP   #$BA
        BCS   NEWDIG
        AND   #$0F
        RTS
MULT10  STA   $07
        LDA   $06
        ASL   A          ；得2A
        ASL   A          ；得4A
        ADC   $06        ；得5A
        ASL   A          ；得10A
        ADC   $07        ；×10+数
        STA   $06        ；存结果到工作单元
        RTS
```

例4和例5两个程序都调用了表2—2中的KEYIN子程序，在例5中NEWDIG和KEYIN构成两重子程序结构。

例6 APPLE Ⅱ 输出码转换程序

目的：将 MPU 的A中八位二进制数转换成三位十进制数的ASCII码，并在显示器上显示（送显示器的字符必须用ASCII码）。

示范题：

a)	A＝FF	（十六进制）
结果：	显示255	（十进制）
b)	A＝80	（十六进制）
结果：	显示128	（十进制）

程序执行前累加器A的值可用监控命令给出。

框图：

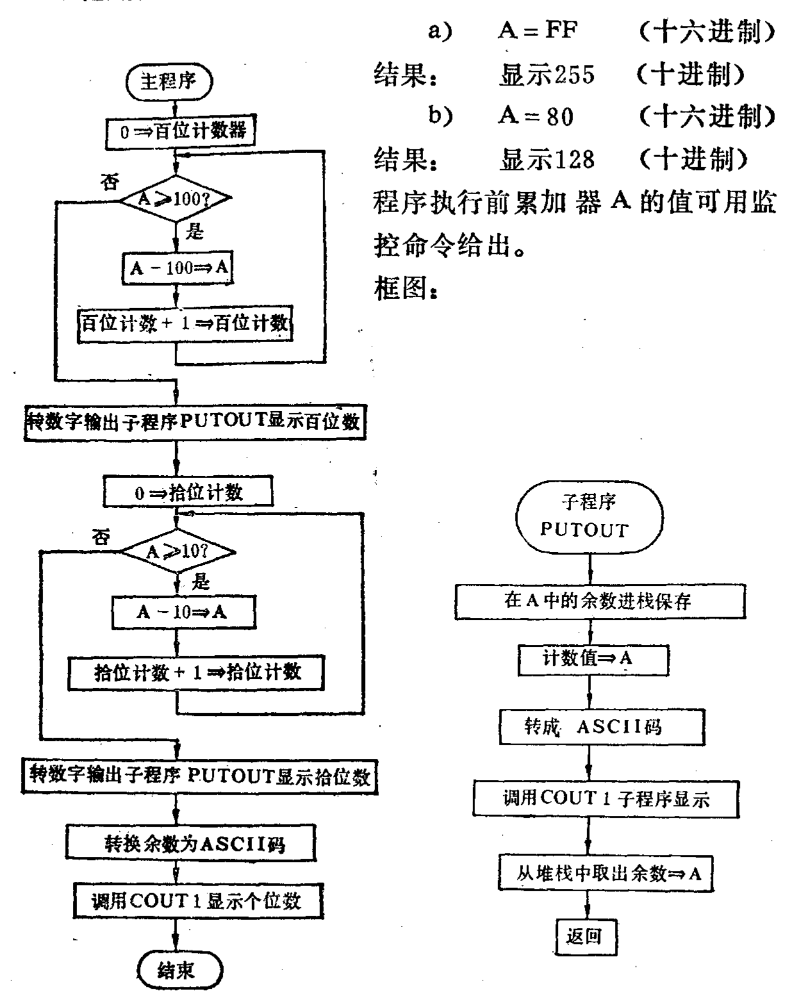


图2—21 APPLE Ⅱ 输出码转换程序框图

源程序：

```
        ORG   $0300
        LDX   #0       ; 0⇒百位计数器
C100    CMP   #100     ; A≥100?
        BCC   OUT1     ; 否，转显示百位数
        SBC   #100     ; 是，A-100⇒A
        INX            ; 百位计数器+1
        JMP   C100     ; 返回再判余数≥100?
OUT1    JSR   PUTOUT   ; 显示百位数
        LDX   #0       ; 0⇒拾位计数器
C10     CMP   #10      ; A≥10?
        BCC   OUT2     ; 否，转显示拾位数
        SBC   #10      ; 是，A-10⇒A
        INX            ; 拾位计数器+1
        JMP   C10      ; 返回再判余数≥10?
OUT2    JSR   PUTOUT   ; 显示拾位数
        ORA   #$B0     ; 余下的个位数转成ASCII码
        JSR   $FDF0    ; 调COUT1显示个位数
        BRK
PUTOUT  PHA            ; 余数进栈保存
        TXA            ; 计数值⇒A
        ORA   #$B0     ; 转成ASCII码
        JSR   $FDF0    ; 调用COUT1显示
        PLA            ; 余数出栈恢复
        RTS
```

此程序调用了表2—2中的COUT1子程序。程序中的PUTOUT和COUT1也构成了双重子程序结构。

这里再分析一下子程序PUTOUT。它在把数据转换成 ASCII

码之前，首先把A累加器的内容保护进栈（称为保护现场），这是因为A中放的是余数，返主之后，主程序还要再接着使用这个余数，因此A中的内容不能由于子程序的执行而丢失。但是子程序PUTOUT也要用A来存放待显示的数，所以就需要先把A保护进栈，然后再执行子程序的内容。此外程序员还必须记住，在子程序内容执行完毕返主之前，必须用出栈指令使恢复A的内容（称为恢复现场），以在返主后供主程序继续使用。

如果主程序和子程序使用的寄存器中，发生冲突的不只是A，还有X、Y、P也发生冲突，那么在子程序中还要加上对X、Y、P的保护现场和恢复现场部分。因此转入子程序时应当首先分析一下是否需要保护现场，若需要，那么是哪几个寄存器需要保护。只有这样仔细的加以处理，才能使程序得以正确执行。

例7 APPLE Ⅱ鸣喇叭程序

目的：使APPLE Ⅱ微型机所配喇叭每隔5秒钟鸣叫一次，共计鸣叫十次。

框图：

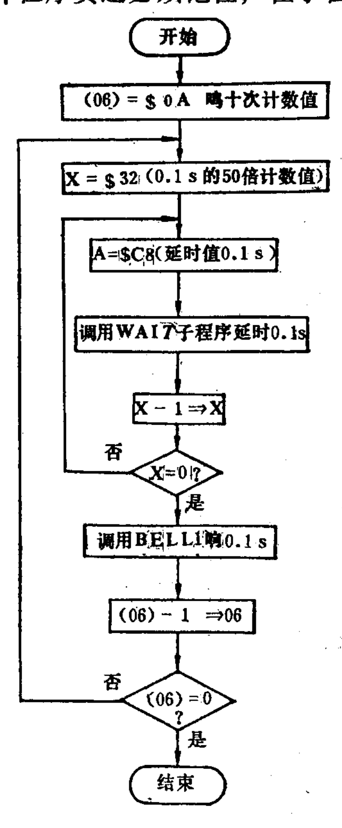


图2—22 APPLE II鸣喇叭程序框图

源程序：

```
WAIT    EQU   $FCA8
BELL1   EQU   $FBDD
        ORG   $0300
```

```
        LDA   #$0A      ; 鸣喇叭次数为10
        STA   $06
LOOP2   LDX   #$32      ; 0.1秒延时值的倍数为50
LOOP1   LDA   #$C8      ; 0.1秒延时值
        JSR   WAIT      ; 调用WAIT产生0.1s延时
        DEX             ; 够5秒否?
        BNE   LOOP1     ; 不够循环
        JSR   BELL1     ; 够,调用BELL1鸣喇叭0.1秒
        DEC   $06       ; 满十次否?
        BNE   LOOP2     ; 否，循环
        BRK             ; 是，结束
```

此程序调用了表2-2中的WAIT子程序以及BELL1子程序。

§2—6 算术运算程序

前面已经介绍过一些简单的算术运算程序，例如二进制及十进制加法、减法程序。在微型机中最基本的算术运算是加法和减法，其它的算术运算例如乘法和除法都是利用加、减及移位操作按照一定的算法组成的指令序列（程序）来实现的，本节将继续介绍一些算术运算程序的编制方法。

例1 多字节十进制数求和

目的：将两个多字节十进制数相加，字节数放在06单元，两个操作数按高位至低位的顺序依次放在6000和6100为起始地址的存贮区域中，运算结果送到6200为起始地址的存贮区域。

示范题：(06) = 04

(6000) = 12　　；8位被加数

(6001) = 83

(6002) = 56

(6003) = 93

(6100) = 24　　　；8位加数

(6101) = 79

(6102) = 12

(6103) = 15

结果：(6200) = 37　　　；8位结果

(6201) = 62

(6202) = 69

(6203) = 08

框图：

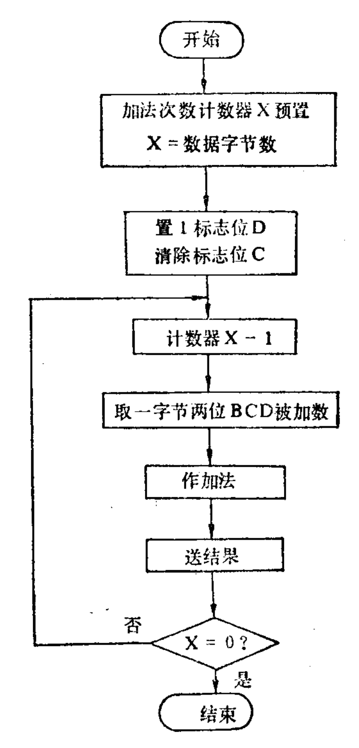


图2—23　8位十进数求和程序框图

源程序：

```
        ORG  $0300
        LDX  $06       ；加法次数预置
        SED            ；BCD加法，D标志置1
        CLC
NEXT    DEX
        LDA  $6000,X   ；从低字节逐次取被加数
        ADC  $6100,X   ；相加
        STA  $6200,X   ；送结果
        TXA            ；判相加次数
        BNE  NEXT      ；次数不够继续加
        BRK            ；已加完，结束
```

此程序中使用了一条TXA指令，目的在于通过 X⇒A的传送影响标志位Z，用以判断X寄存器内容是否已为0值。

例2 求二进制数的平方根

目的：将存放在XSQ单元中的数X，求得其平方根 $\sqrt{X}$ 后存放到0006单元中，为简单起见设X为整数，方根值也取整数。

示范题：a) (XSQ) = 19　　此为十六进制数19即十进制数25

结果：(06) = 05

b) (XSQ) = 65　　此为十六进制数65即十进制数101

结果：(06) = 0A

求方根方法：我们知道任何正整数都有如下的性质：

$1^2 = 1$

$2^2 = 1 + 3$

$3^2 = 1 + 3 + 5$

⋮

⋮

$$N^2 = \underbrace{1+3+5+\cdots\cdots+(2N-1)}_{N\text{个奇整数}}$$

所以可以把一个正整数 $X=N^2$ 连续逐次减去奇数1、3、5、……$(2N-1)$ 直到结果为0或不够减时为止。所减去奇整数的个数恰好就是这个正整数的整数平方根。

框图：

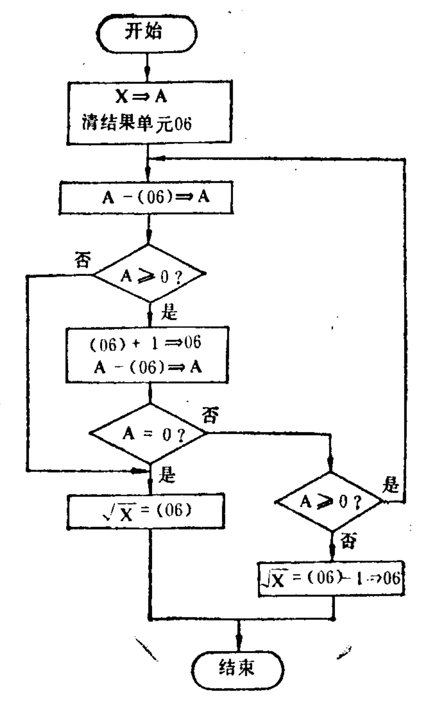


图2—24 求二进制数平方根程序框图

源程序：

```
XSQ   EQU   $6000
      ORG   $0300
```

```
      CLD
      LDA   XSQ    ；取X到A中
      LDY   #00
      STY   $06    ；清结果单元06
      SEC
LOOP  SBC   $06
      BCC   DONE   ；不够减转结束，结果在06单元
      INC   $06    ；(06)为已减的奇整数个数
      SBC   $06    ；和LOOP处指令合起来正好减去
                     了奇整数
      BEQ   DONE   ；减结果为0,转结束,结果在06单元
      BCS   LOOP   ；减结果>0，继续减
      DEC   $06    ；减结果<0，则从结果中扣除1
DONE  BRK
```

例3 8位二进制整数乘法

目的：将存放在存贮单元 MPD 中的二进制被乘数和存放在存贮单元MPR中的二进制乘数相乘，乘积高8位存放在 RESUH 单元乘积低8位存放在RESUL单元。

示范题：a) (MPD) = 6F

(MPR) = 61

结果 (RESUH) = 2A

(RESUL) = 0F

即6F×61 = 2A0F　写成十进制就是

111×97 = 10767

b) (MPD) = 76

(MPR) = 5B

结果 (RESUH) = 29

(RESUL) = F2

即76×5B＝29F2　写成十进制就是：

118×91＝10738

方法：采用部分积逐次右移的一位乘法即每次由乘数最低位代码决定部分积是否要和被乘数相加。然后把部分积右移1位，右移出的低位代码逐次移进乘积低8位结果单元中。于是两个8位二进制数相乘所得到的乘积则为十六位数。下面以两个4位数相乘得到8位乘积的运算为例说明此法：

	乘数	被乘数	部分积
	1 1 0 1	1 1 1 1	0 0 0 0
1）	乘数最低位为1，加被乘数		+ 1 1 1 1
	部分积右移一位		1 1 1 1
			0 1 1 1 1
2）	乘数为0，不加被乘数		0 0 1 1 1 1
	部分积右移一位		
3）	乘数为1，加被乘数		+ 1 1 1 1
	部分积右移一位		1 0 0 1 0 1 1
			1 0 0 1 0 1 1
4）	乘数为1，加被乘数		+ 1 1 1 1
	部分积右移一位		1 1 0 0 0 0 1 1
			1 1 0 0 0 0 1 1

由算式的结果可以看到，在部分积位置上得到的是高位积1100，每次右移出的码构成了低位积0011，一共是8位乘积，由于部分积每次右移出去的数不再参加运算，所以参加相加操作的只有四位。同理此法用于8位数相乘时，可得16位乘积，但参加操作的只有8位，所以运算器就可采用8位加法器。

框图：

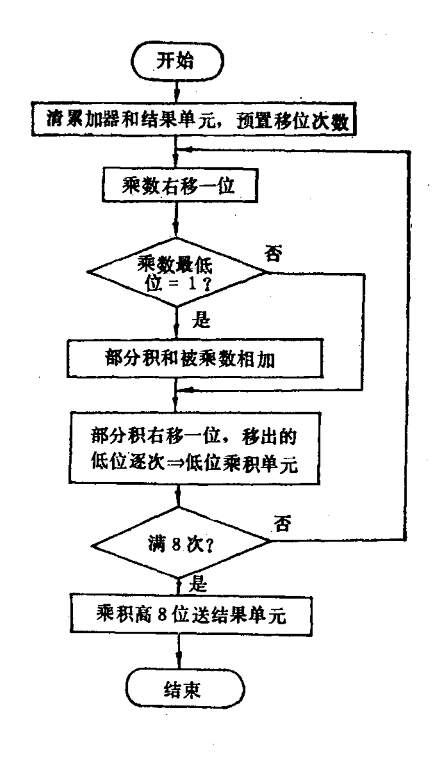


图2—25　8位乘法程序框图

源程序：

```
MPD   EQU   $6000        ；被乘数存放单元
MPR   EQU   MPD+1        ；乘数存放单元
```

```
RESUH   EQU  MPR+1      ; 高位积存放单元
RESUL   EQU  RESUH+1    ; 低位积存放单元
TEMP    EQU  $06        ; 工作单元
        ORG  $0300
MULT    CLD
        LDA  MPR        ; 转存乘数以防移位后丢失乘数
        STA  TEMP
        LDA  #00        ; 清存放部分积的累加器
        STA  RESUL      ; 清结果单元
        STA  RESUH
        LDX  #08        ; 预置移位次数
LOOP    LSR  TEMP       ; 判乘数最低位代码
        BCC  NOADD      ; 是0，转NOADD
        CLC             ; 是1，部分积和被乘数相加
        ADC  MPD
NOADD   ROR  A          ; 部分积右移1位，相加产生的
                        进位C移进最高位
        ROR  RESUL      ; 移出的低位送入RESUL单元
                        高位
        DEX             ; 满8次否
        BNE  LOOP       ; 不满，再继续循环
        STA  RESUH      ; 满，把乘积高8位送入
                        RESUH单元
        BRK
```

例4　16位二进制整数乘法

目的：将存放在存贮单元MPDH及MPDL中的二进制被乘数和存放在存贮单元MPRH及MPRL中的二进制乘数相乘、乘积为32位，依由高位向低位次序，顺序放入存贮单元RES1，RES2，

RES3，和RES4中

示范题：a）(MPDH) = 04

(MPDL) = 00

(MPRH) = 04

(MPRL) = 00

结果 (RES1) = 00

(RES2) = 10

(RES3) = 00

(RES4) = 00

即0400×0400 = 00100000（十六进制）

写成十进制就是 $2^{10}\times2^{10}=2^{20}$

b）(MPDH) = 20

(MPDL) = 00

(MPRH) = 20

(MPRL) = 00

结果 (RES1) = 04

(RES2) = 00

(RES3) = 00

(RES4) = 00

即 2000×2000 = 04000000（十六进制）

写成十进制就是 $2^{13}\times2^{13}=2^{26}$

方法：同于八位乘法，但此处乘数移位时要安排成16位右移，部分积移位时要安排成32位右移。

源程序：

```
MPDH   EQU   $6000   ; 被乘数高8位存放单元
MPDL   EQU   $6001   ; 被乘数低8位存放单元
MPRH   EQU   $6002   ; 乘数高8位存放单元
MPRL   EQU   $6003   ; 乘数低8位存放单元
```

```
TEMH   EQU   $6004   ; 工作单元
TEML   EQU   $6005   ; 工作单元
RES1   EQU   $6006   ; 高位积存放单元
RES2   EQU   $6007   ; 高位积存放单元
RES3   EQU   $6008   ; 低位积存放单元
RES4   EQU   $6009   ; 低位积存放单元
       ORG   $0300
       LDA   MPRH    ; 转存乘数，防止乘数丢失
       STA   TEMH
       LDA   MPRL
       STA   TEML
       LDA   #00     ; 累加器A清0
       LDY   #$10    ; 移位次数预置
       LDX   #00     ; 变址计数器预置
CLEAR  STA   RES1,X  ; 四个结果单元清0
       INX
       CPX   #04
       BNE   CLEAR
LOOP   LSR   TEMH    ; 判乘数最低位代码
       ROR   TEML
       BCC   NOADD   ; 为0，转NOADD移位
       CLC           ; 为1，部分积和被乘数相加
       LDA   RES2
       ADC   MPDL
       STA   RES2
       LDA   RES1
       ADC   MPDH
       STA   RES1
```

```
NOADD  ROR  RES1   ；部分积右移一位
       ROR  RES2
       ROR  RES3
       ROR  RES4
       DEY         ；移位次数够16次否？
       BNE  LOOP   ；不够，循环
       BRK         ；够，结束
```

例6 八位二进制整数除法

目的：将存放在DVNH单元和 DVNL单元中的16位二进制被除数，除以存放在DVS单元中的8位二进制除数，商取8位存放在QUOT单元，余数在累加器A中

示范题：(DVNH) = 6D 被除数高8位

(DVNL) = 3A 被除数低8位

(DVS) = 76 除数

结果(QUOT) = EC 商

(累加器A) = 72 余数

即 6D3A ÷ 76 = EC（余数72）

写成十进制就是 27962 ÷ 118 = 236（余数114）

由于本程序针对八位二进制整数除法，所以要求16位被除数，8位除数，并且要求被除数高8位小于除数，这样就可得到8位商。

在介绍除法程序之前，我们先来介绍一下计算机进行除法运算的方法。我们知道，人在进行二进制笔算除法时是每次将余数（第一次是将被除数）和除数比较一下，够减时就减，且上商1，不够减时就不减，且上商0，然后再求下位商。在用余数减去除数时，要先将除数右移一位。我们可以举一个笔算除法的例子来分析一下：

011001 ÷ 101

```
          1 0 1
      ____________
101) 0 1 1 0 0 1
       1 0 1
      ____________
           1 0 1
           1 0 1
          ________
               -
```

可见笔算除法是借助于减法（余数－除数）和移位（除数右移）来完成的，而且在做减法时要先判断一下，够减才减，不够减就不减。但是用计算机进行除法运算可以有不同的运算方法。例如恢复余数法，这种方法也是采用减法操作和移位操作来完成除法运算，但它和笔算除法有以下两点区别：

1．笔算除法每次相减时是保持余数位置不动，而将除数右移一位。恢复余数法每次相减时，则是保持除数位置不动，而将余数左移一位。

2．在作减法时，和笔算除法不同，恢复余数法是不管够减不够减，先减了再说，减了之后再根据标志位来判断这次是否够减，如果够减就上商1，再继续求下位商。如果不够减，说明余数是个负数，则应重新加上除数，以恢复这次相减之前余数的原值，称为恢复余数，并且上商0，然后再继续求下位商。

我们也举一个例子来说明恢复余数法的运算方法。

例 被除数A＝6D3A

除数B＝76　　（补码为8A）

商　＝EC

余数　＝72

	0 1 1 0 1 1 0 1	0 0 1 1 1 0 1 0	商
←	1 1 0 1 1 0 1 0	0 1 1 1 0 1 0 0	
－除数	＋1 0 0 0 1 0 1 0		1 1 1 0 1 1 0 0
C＝1够减	0 1 1 0 0 1 0 0		
←	1 1 0 0 1 0 0 0	1 1 1 0 1 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝1够减	0 1 0 1 0 0 1 0		
←	1 0 1 0 0 1 0 1	1 1 0 1 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝1够减	0 0 1 0 1 1 1 1		
←	0 1 0 1 1 1 1 1	1 0 1 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝0不够减	1 1 1 0 1 0 0 1		
恢复余数	＋0 1 1 1 0 1 1 0		
	0 1 0 1 1 1 1 1		
←	1 0 1 1 1 1 1 1	0 1 0 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝1够减	0 1 0 0 1 0 0 1		
←	1 0 0 1 0 0 1 0	1 0 0 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝1够减	0 0 0 1 1 1 0 0		
←	0 0 1 1 1 0 0 1	0 0 0 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝0不够减	1 1 0 0 0 0 1 1		
恢复余数	＋0 1 1 1 0 1 1 0		
	0 0 1 1 1 0 0 1		
←	0 1 1 1 0 0 1 0	0 0 0 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝0不够减	1 1 1 1 1 1 0 0		
恢复余数	＋0 1 1 1 0 1 1 0		
	0 1 1 1 0 0 1 0		

由于恢复余数法中包括恢复余数的操作，所以用这种方法来编制除法程序时，执行速度就比较慢。为了取消恢复余数的步骤，可以改用加减交替法来作除法，则更为理想。怎样才能取消恢复余数的操作呢？我们可以再来仔细分析一下恢复余数法的步骤。

若上步相减结果所得余数为R_i，则求本位商的操作为：

⋮

余数左移－除数　$2R_i - B = R_{i+1} < 0$　余数为负，本位商0

＋除数以恢复余数　$R_{i+1} + B = 2R_i - B + B = 2R_i$

余数左移－除数　$2(2R_i)-B=4R_i-B$　若余数为正，则下位商1

求下位商

若余数为负，则下位商0

⋮

为了取消恢复余数操作，可将这种算法改为以下所示的算法

⋮

余数左移－除数　$2R_i-B=R_{i+1}<0$　余数为负，本位商0

余数左移＋除数　$2R_{i+1}+B=2(2R_i-B)+B$

求下位商　$=4R_i-B$　若余数为正，则下位商1

若余数为负，则下位商0

⋮

可见两种方法所得结果是相同的，这后一种方法叫做加减交替法，它的要领为：

本次相减所得余数为正（够减）时商1，且下位求商操作为余数左移一位后减除数，若本次相减所得余数为负（不够减）时，商0，则下位求商操作为余数左移一位后加除数。

我们仍用上例中的除数、被除数为例写出加减交替法的操作过程。通过具体例子的说明帮助我们加深对加减交替法的了解。

例　被除数＝6D3A

除数＝76　（补码为8A）

商＝EC

余数＝72

	0 1 1 0 1 1 0 1	0 0 1 1 1 0 1 0	商
－除数	＋1 0 0 0 1 0 1 0		
C＝0	1 1 1 1 0 1 1 1		0
←	1 1 1 0 1 1 1 0	0 1 1 1 0 1 0 0	
＋除数	＋0 1 1 1 0 1 1 0		
C＝1	0 1 1 0 0 1 0 0		0 1
←	1 1 0 0 1 0 0 0	1 1 1 0 1 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝1	0 1 0 1 0 0 1 0		0 1 1
←	1 0 1 0 0 1 0 1	1 1 0 1 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝1	0 0 1 0 1 1 1 1		0 1 1 1
←	0 1 0 1 1 1 1 1	1 0 1 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝0	1 1 1 0 1 0 0 1		0 1 1 1 0
←	1 1 0 1 0 0 1 1	0 1 0 0 0 0 0 0	
＋除数	＋0 1 1 1 0 1 1 0		
C＝1	0 1 0 0 1 0 0 1		0 1 1 1 0 1
←	1 0 0 1 0 0 1 0	1 0 0 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝1	0 0 0 1 1 1 0 0		0 1 1 1 0 1 1
←	0 0 1 1 1 0 0 1	0 0 0 0 0 0 0 0	
－除数	＋1 0 0 0 1 0 1 0		
C＝0	1 1 0 0 0 0 1 1		0 1 1 1 0 1 1 0
←	1 0 0 0 0 1 1 0	0 0 0 0 0 0 0 0	
＋除数	＋0 1 1 1 0 1 1 0		
C＝0	1 1 1 1 1 1 0 0		1 1 1 0 1 1 0 0
恢复余数	＋0 1 1 1 0 1 1 0		
	0 1 1 1 0 0 1 0		
	余数		

以上求商运算要求被除数和除数的最高位为0，否则在求商过程中，余数左移时将可能造成最高位有效数值的丢失而使运算发生错误。为此，本程序在转入求商之前，先对被除数及除数的最高位是否为1进行判别，若为1则将被除数及除数各乘以1/2然后再转入求商，当然这可能对精度略有影响。

框图：

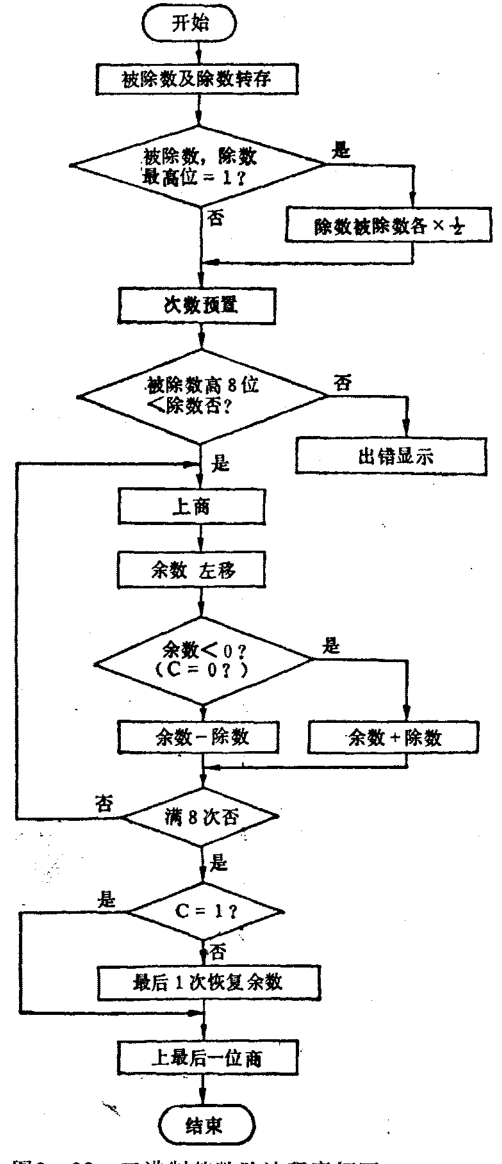


图2—26　二进制整数除法程序框图

源程序:

```
DVNH   EQU   $6000  ；被除数高8位所在单元
DVNL   EQU   $6001  ；被除数低8位所在单元
DVS    EQU   $6002  ；除数所在单元
QUOT   EQU   $6003  ；商所在单元
TMPH   EQU   $6004  ；工作单元
TMPL   EQU   $6005  ；工作单元
TMPS   EQU   $6006  ；工作单元
       ORG   $0300
       LDA   DVNH   ；将被除数及除数转存
       STA   TMPH
       LDA   DVNL
       STA   TMPL
       LDA   DVS
       STA   TMPS
       LDA   DVNH   ；测被除数最高位是1否?
       BPL   TEST   ；否，转去测除数最高位
       JMP   SHIFI  ；是,转除数、被除数×1/2
TEST   LDA   DVS    ；测除数最高位为1否?
       BPL   START  ；否，转求商
SHIFT  LSR   TMPS   ；是，除数、被除数×1/2
       LSR   TMPH
       ROR   TMPL
START  CLD
       SEC
       LDY   #$08   ；次数预置
```

```
      LDA  TMPH
      SBC  TMPS    ；判被除数高8位<除数否？
      BCS  ERR     ；否，不能除，转出错显示
LOOP  PHP          ；是，保存标志位C
      ROL  QUOT    ；上商，C=0商0，C=1商1
      ASL  TMPL    ；16位余数联合左移1位
      ROL  A
      PLP          ；恢复标志位C
      BCC  ADD     ；余数为负，转ADD加除数
      SBC  TMPS    ；余数为正，减除数
      JMP  NEXT
ADD   ADC  TMPS
NEXT  DEY          ；够8次否？
      BNE  LOOP    ；不够，继续循环
      BCS  LAST    ；够，判最后一次余数，若为正，
                     转LAST
      ADC  TMPS    ；最后一次余数为负，恢复余数
      CLC
LAST  ROL  QUOT    ；上最后一位商
      BRK
ERR   LDA  #$C5    ；调用COUT1在显示器上显示ERR字
                     符
      JSR  $FDF0   ；C5为E的ASCII码
      LDA  #$D2    ；D2为R的ASCII码
      JSR  $FDF0
      LDA  #$D2
      JSR  $FDF0
      BRK
```

需要说明一下，此程序只适用16位被除数，8位除数，求得8位商的整数除法情况。若被除数只有8位则应将被除数置于低8位而在高8位补上8个0，并应使被除数低8位大于除数。若进行8位二进制小数除法，则除数、被除数都应为8位小数，且被除数应小于除数才能使商为8位小数，在编制相应程序时必须将这些问题考虑在内。

§2—7 输入/输出工作方式及6502的中断系统

本章以上各节涉及到的内容主要是针对微处理机6502及存贮器两部分。而一个由6502构成的微型计算机系统除了6502及存贮器外，还有一个不可缺少的组成部分——输入/输出设备（简称I/O设备）。例如作为输入设备的键盘，作为输出设备的显示器都是构成微型机不可缺少的。此外，还有各种打印机、盒式磁带及软磁盘，CRT显示器等等也是微型机常用的输入/输出设备。当微型机用于工业控制和数据采集系统时，常常要用到的模数转换器(A/D)和数模转换器(D/A)也都是属于输入/输出设备之列的。总之，输入/输出设备种类繁多，这里只是略举几例来说明它们对于微型机应用的重要性。那么输入/输出设备是如何和6502微处理机进行信息交换的呢？我们不妨回顾一下存贮器和6502之间是如何进行信息交换的，其实这两类问题是很相似的。由6502通过地址总线$A_0 \sim A_{15}$向存贮器发出地址码（例如用符号地址N1来表示此地址）并向存贮器芯片发出R/W＝1的控制信号，则N1地址单元的内容就会由数据总线$D_0 \sim D_7$送往6502，这就是读存贮器的过程，即LDA N1指令的执行过程。而6502要由输入设备取回一个数时，其过程大体与上面相似，即由6502通过$A_0 \sim A_{15}$向一个指定的输入设备发出该设备的地址码（例如用符号地址N2来表示）并发出R/W＝1的控制信号，则该输入设备中的数据就可通

过数据总线$D_0 \sim D_7$送进6502，这也就是LDA　N2指令的执行过程。但是应当注意，这个地址码N2是一个输入设备的地址码而不是存贮器的地址码，因此在硬件连接上，这个地址N2上接的是输入设备，而没有接存贮器，也就是说存贮器不能用已被输入设备占用了的地址码 N2，对于 6502向存贮器输出一个数和 6502向输出设备输出一个数的过程，我们也可以作同样的类比。这种MPU对I/O的寻址方式称为存贮器对应输入/输出方式（或称存贮器映象方式）。它的特点是I/O设备的地址码和存贮器的地址码是统一编址的，也就是说 $A_0 \sim A_{15}$ 16 根地址线所包括的64K地址容量中，要开辟出一部分让给 I/O 设备用。这会使得存贮器所占的容量减少了，例如若是辟出8K地址范围作为 I/O设备的地址码用，那么存贮器就只能使用56K 地址范围了。但是所有存贮器可以使用的指令(例如LDA，STA，INC，DEC，BIT等等）都可以用于I/O 设备，而不必另设I/O设备专用的指令，这就给对I/O设备的编程带来极大的方便。

另外还有一种MPU 对 I/O的寻址方式称为专用输入/输出方式(或称为端口映象方式)，例如 Z80微处理机采用的就是专用输入/输出方式，关于这种方式我们此处就不进行讨论了。

可是当我们深入一步来研究输入/输出设备的具体工作情况时，就会发现处理输入/输出设备的问题要比处理存贮器复杂得多，这是因为各种存贮器芯片构成的存贮器，它们的规格和使用方法都是大体相同的,工作速度也几乎和微处理机一样,一片存贮器芯片内部集中了大量的存贮单元（以K为单位)及有关的控制电路，因此在硬件连接上也比较简单,除了地址码和读/写控制信号外一般不需别的控制信号，而输入/输出设备却不同，它们的种类繁多,无论从构造、工作速度或是从数据传送方式上都有很大的差异，所以一般说来各个I/O设备都要通过“接口”接入微型机。微处理机和I/O设备之间的数据交换要经过这个中间电路——“接口”

来进行。6502和接口之间的数据交换用指令来进行，所以I/O地址码(例如前面举的例中用的N2)是设在接口上的。而接口和I/O设备之间的数据交换不能用指令进行，可用硬件使它们直接传递数据或是用联络信号来实现。

考虑到I/O设备工作速度，数据传送方式的差异以及如何充分发挥MPU的效率等问题，I/O设备和微型机之间的数据传送可有以下四种方式

1．无条件传送方式(同步传送方式)

2．查询方式

3．中断方式

4．DMA方式(直接存贮器访问方式)

本书主要介绍前三种方式。现在依次介绍如下：

一、无条件传送方式

所谓无条件是指输入/输出设备对于数据传送时间的要求或者是固定时间的或者是没有特别要求而由程序执行的时间来决定，这是最简单的一种I/O数据传送方式。硬件、软件都很节省。图2—27是无条件输入传送的一个电路举例。

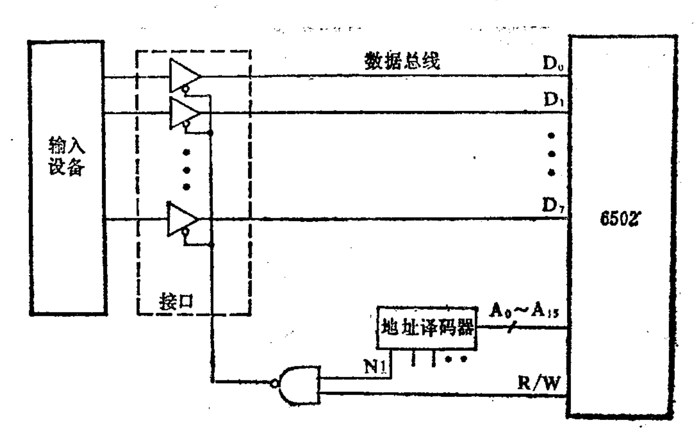


图2—27　无条件输入传送电路

这里为了实现输入传送，在输入设备和6502之间使用了一组三态门作为接口电路，接口的地址码选定为N1(符号地址)，那么只要在程序中执行一条LDA N1的指令就实现了将输入设备中的数据送入累加器A的无条件传送。但是用这种方式进行输入传送时，输入设备中的数据必须是处于已准备好发送的状态，那么6502只要接收数据就行了。例如当八个开关的状态作为输入设备接入上述电路时，就可以用LDA N1指令测试出这些开关中哪些是合上的，哪些是断开的。注意，在输入传送中用的接口是不能带数据锁存的，否则这些数据将长时间挂在数据总线上而破坏了整个系统的工作，因此这里用的是三态门作为输入接口，当LDA N1指令执行后，三态门就处于高阻状态而使输入数据和数据总线断开。

下面我们再看一个无条件输出传送的电路举例，如图2—28所示。

这里使用了一条锁存器作为输出接口，其地址码选为N2，那么只要在程序中执行一条STA N2的指令就实现了将累加器A中的数据打入到锁存器中，再传送给输出设备，但是用这种方式进行输出传送时，输出设备必须是已处于准备好接收数据的状态，

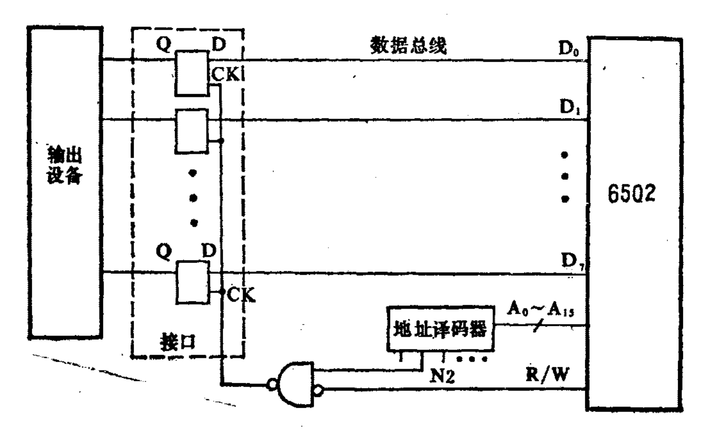


图2—28 无条件输出传送电路

那么6502就只要发送数据就行了。例如当用显示灯作为输出设备时，就可以用这种传送方式。

还应当指出，输出的数据是应当带锁存的。因为输出设备不能只对一条STA　N2指令执行中哪几个微秒时间内传送过来的数据起作用，所以输出接口用的是锁存器而不是三态门。

顺带说一句，我们这里介绍的“接口”都是最简单的电路，实际上6502配有专门的接口芯片供它和I/O设备连接时使用。这些专门的接口芯片都是功能很强的可编程接口芯片。在第四章中将向大家介绍这些接口芯片的使用方法。

二、查询方式

一般说来，I/O设备的工作速度是不能和MPU 相比较的，它要比MPU的速度慢得多，所以它们之间的数据交换常常采用异步的方式进行。例如对于键盘这样的输入设备，6502并不是什么时候都可以从键盘取得数据的，而必须等有键被按下产生了一个稳定的数据之后，才能让6502取走这个数，所以输入设备键盘除了要向6502提供数据信息外，还要向6502提供状态信息 READY，

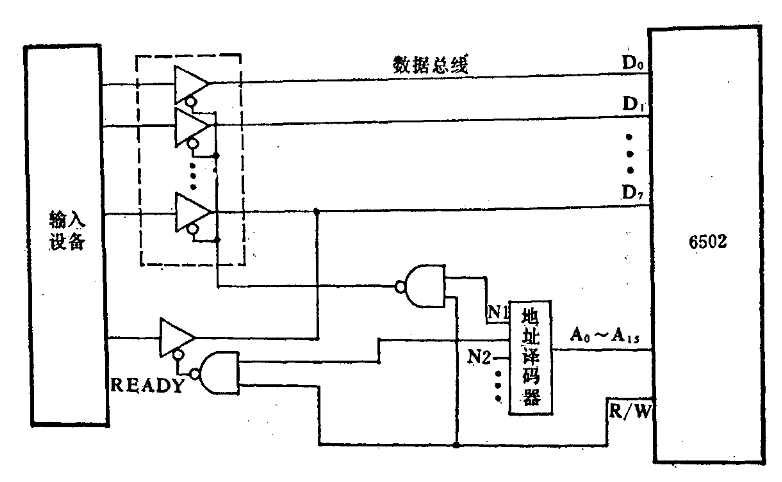
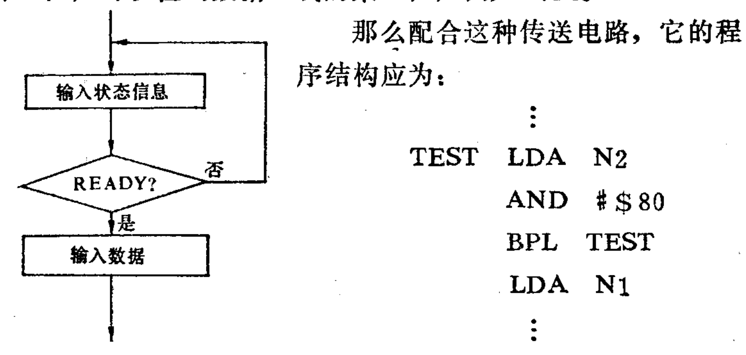


图2—29　查询式输入传送电路

当READY＝1时，表示键盘有数可取；READY＝0时表示键盘中数尚未准备好，因此6502在从键盘取数之前必须首先查询一下READY的状态以决定能否取数，这种查询方式输入传送的电路，可由图2—29示意。

如图2—29所示，数据信息传送用N1地址码，而状态信息传送用N2地址码，即接口电路使用了两个地址码。而且此处状态信息只有一位，可以挂到数据总线的某一位，例如D_7上。



那么配合这种传送电路，它的程序结构应为：

```
        ⋮
TEST  LDA  N2
      AND  #$80
      BPL  TEST
      LDA  N1
        ⋮
```

这就完成了一次查询式数据输入。

下面我们再来看查询式输出传送方式的工作过程，例如对于打印机这样的输出设备，它并不是什么时候都可以接收来自6502的数据的，当它正在打字时是不能接收数据的，只有打印完一个字符时才能再接收下一个字符。因此打印机除了接收6502的数据信息外，还需向6502提供一个状态信息BUSY，当BUSY＝1时表示正在打印字符，不能接收数据；当BUSY＝0时，表示前一字符已经打印完毕，可以接收下一字符的数据了。此时6502才能向打印机输出字符，因此6502在向打印机输出数据之前，必须先查询一下BUSY的状态以决定能否送数，这种查询式输出电路可用图2—30来示意。

如图2—30所示，数据传送用N3地址码，状态信息传送用N4地址码，即接口电路使用了两个地址码，数据信息传送是输出方

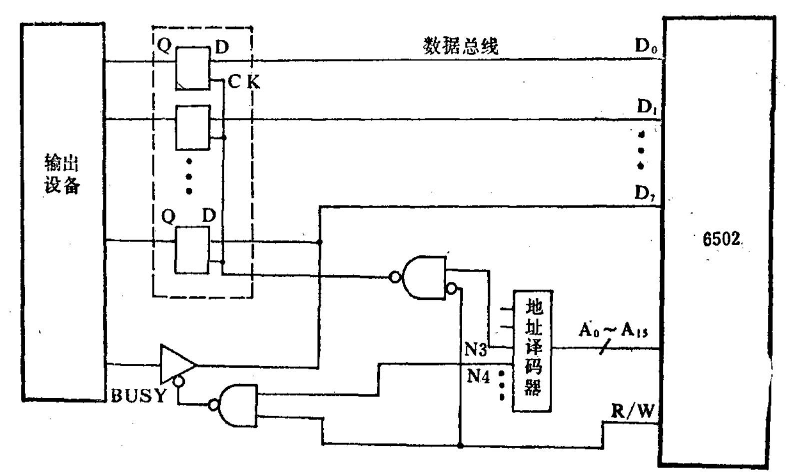
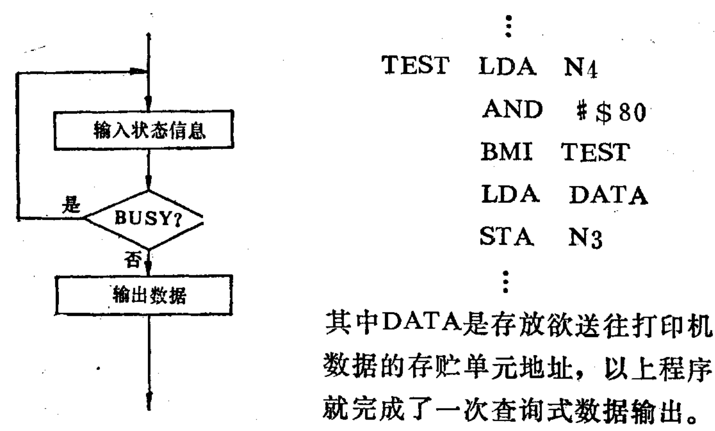


图2—30　查询式输出传送电路

式，状态信息传送是输入方式，也只有一位状态信息BUSY，我们也把它挂在数据总线D_7上。那么配合这种传送电路，它的程序结构应为：




```
        ⋮
TEST  LDA   N4
      AND   #$80
      BMI   TEST
      LDA   DATA
      STA   N3
        ⋮
```

其中DATA是存放欲送往打印机数据的存贮单元地址，以上程序就完成了一次查询式数据输出。

关于查询式输入/输出传送方式此处只是作一些初步介绍，在第四章及第五章中还要结合接口芯片的使用作详细和深入地介绍。

三、中断方式

使用查询方式，当外设未准备好时MPU要不断地查询外设，例如上面举例用的两段查询程序中的头三条，它们要执行许多遍循环之后，才能得到允许交换数据的状态信息（READY＝1 或BUSY＝0），程序才能向下滑行，以取回输入设备的数据或是送出一个数据给输出设备。反复查询I/O设备的过程就是6502微处理机在等待I/O设备，而不能做别的事情，这就浪费了 MPU6502的宝贵时间，高速的MPU 让低速的I/O设备拖住了后腿，而不能充分发挥出 MPU 可以高速度工作的优势。要知道反复查询的这段时间，对于6502来讲是足够执行许多条指令的。另外，采用查询式传送方式难以同时启动多台I/O设备工作，这就是中断传送方式产生的原因。可以设想一下，I/O设备允许交换数据的READY或BUSY状态信息能否不由MPU查询而是由I/O设备自己向MPU 发出呢？这样I/O设备在它内部处理数据（例如等待按键或是正在打字）而不能和MPU交换数据时，MPU就可以去干别的事情（执行其它程序），而一旦I/O设备内部处理数据完毕（例如已有键按下或是上一个字符已经打印完毕）要求和MPU交换数据时就向MPU发出一个中断请求信号，要求MPU响应它的中断请求，暂时中断一下正在执行的程序（称为主程序）转过来同要求中断的 I/O 设备交换数据，即为要求中断的I/O设备服务（称为执行中断服务程序），等到服务完毕时再返回刚才被中断的主程序，继续执行下去。这就是中断传送方式的基本思想。为了说明这件事，我们可以打个比喻，这好象一个商店的经理正在结算帐目，突然来了一位顾客，请求购买物品，经理立即放下正在结帐的工作，而去为顾客服务。只有送走了顾客，他才能返回来继续刚才被打断的结帐工作。这个过程同计算机的中断请求及中断响应的过程极为相似。中断是计算机中一种最重要的输入/输出传送方式。

为了实现中断传送方式，6502的中断系统必须具有以下功能：

1. 为了使外设能提出中断请求，必须在接口中设置中断请求触发器。而且为了能灵活选择是否让外设进入中断方式，要在接口中设置中断屏蔽触发器。这样，即使外设有中断请求也不一定能够允许它向6502发出，这就靠控制接口中的中断屏蔽触发器的状态来完成。

2. 对于外设提出的中断请求，6502要具有响应中断的功能，而且为了灵活起见，应当使6502对于已提出的中断请求$\overline{IRQ}$是否响应具有选择的能力。这是用6502内部标志寄存器P中的中断禁止标志位I来实现的。如果标志位I＝0，则6502收到中断请求$\overline{IRQ}$后，等到当前正在执行的一条指令执行完毕时，就自动转入中断响应周期。如果I＝1，则对于外设已提出的中断请求，6502不予理睬，不作响应，而继续执行正在执行的主程序。标志位I的状态可由程序员用CLI，SEI指令来控制。

3. 6502中断系统还必须具有中断返回功能，即如果6502响应中断，则经过中断响应周期后，6502就由执行主程序而转向执行中断服务程序。当中断服务程序执行完毕时，返回主程序这件事是由在中断服务程序的末尾放上一条中断返回指令RTI来实现的，这点和子程序末尾要放一条子程序返回指令RTS的目的是相同的。

4. 要能实现中断优先权排队。采用中断方式时可以有多台I/O设备同时工作，这样就有可能出现两个或两个以上的I/O设备（称为中断源）同时提出中断请求。那么6502到底响应哪个中断源的中断请求呢？这就必须对各个中断源确定一个中断优先权级别，当多个中断源同时提出中断请求时，6502能找到优先权级别最高的中断源而对它的中断请求予以响应，当优先权级别最高的中断请求处理完毕后再去响应级别较低的中断请求。这个中断优先权排队可由硬件来实现，也可由软件来实现。

下面我们来看一下6502的引腿图就可知道6502中断请求分为

两类，一类是可屏蔽中断请求$\overline{\text{IRQ}}$，另一类是不可屏蔽中断请求$\overline{\text{NMI}}$。上面所谈主要是针对可屏蔽中断请求$\overline{\text{IRQ}}$。这两条中断请求的输入线平时都应是高电平，只有当它们有中断请求时才变为低电平。

这两种中断请求有什么区别呢？——可屏蔽中断请求$\overline{\text{IRQ}}$是可以被接口电路中的屏蔽触发器所屏蔽而不发到6502来，而且对于即使已经送到6502的中断请求$\overline{\text{IRQ}}$还可以由6502中的中断禁止标志位I的状态来决定是否响应。但是不可屏蔽中断请求$\overline{\text{NMI}}$就不同，$\overline{\text{NMI}}$请求一经提出，6502就必须响应，而不能禁止。$\overline{\text{NMI}}$请求的级别要比$\overline{\text{IRQ}}$请求的级别高。

现在我们分别详细介绍一下这两种中断请求的响应过程。首先看一下$\overline{\text{NMI}}$的中断响应，只要$\overline{\text{NMI}}$线上出现负跳变，就将会被6502所识别（要求$\overline{\text{NMI}}$线保持低电平至少2μs）。由于它是不可屏蔽中断请求，因此6502在执行完正在执行的本条指令之后就一定会暂停主程序而进入$\overline{\text{NMI}}$中断响应周期。在$\overline{\text{NMI}}$中断响应周期中6502将自动进行以下工作：

（1）将程序计数器PC的值（即主程序断点的值）送入堆栈保存，先将PC的高字节进栈，再将PC的低字节进栈。

（2）接下来是将标志寄存器P的内容送入堆栈保存。

（3）将标志寄存器P的中断禁止位I置1，禁止再响应其它中断。

（4）6502从地址码为FFFA和FFFB的两个存贮单元中取出地址码（前者为低八位，后者为高八位）送PC，因此$\overline{\text{NMI}}$中断响应周期结束之后，6502将从PC中放进的新地址开始运行。一般说来，FFFA和FFFB单元中往往存放的就是不可屏蔽中断服务程序的入口地址，因此可以把FFFA和FFFB两单元称为指向中断服务程序入口的中断向量。

以上就是不可屏蔽中断响应过程。应当注意，不可屏蔽中断

响应是指对引腿$\overline{\text{NMI}}$上的负跳沿作出响应。

而可屏蔽中断请求$\overline{\text{IRQ}}$的响应过程（如果允许响应的话）很类似不可屏蔽中断$\overline{\text{NMI}}$响应过程，不过它是对引腿$\overline{\text{IRQ}}$上的低电平进行响应，而不是对负跳沿进行响应。在$\overline{\text{IRQ}}$中断响应周期中，6502将自动进行以下工作：

（1）将程序计数器PC的值（即主程序断点的值）送入堆栈保存，先送PC高字节，再送PC低字节。

（2）接下来是将标志寄存器P的内容送堆栈保存。

（3）将标志寄存器P的中断禁止标志位I置1，禁止再响应其它I/O设备提出的$\overline{\text{IRQ}}$请求（但不能禁止响应$\overline{\text{NMI}}$中断请求）。

（4）6502从地址为FFFE和 FFFF两个存贮单元中取出地址码（前者为低八位，后者为高八位）送PC，因此$\overline{\text{IRQ}}$中断响应周期结束后，6502将从PC中放进的新地址开始运行。一般说来，FFFE和FFFF两个单元中往往放的就是中断服务程序入口地址，亦可称它为指向服务程序的中断向量。

由上述的内容可以看到，$\overline{\text{IRQ}}$中断响应和$\overline{\text{NMI}}$中断响应的差别仅在于存放中断向量的存贮单元不同，前者用的是FFFE和FFFF两单元，而后者使用的是FFFA和FFFB两单元。

下面我们再介绍一下中断服务程序编写的一般模式。所谓一般模式就是指这里并不涉及具体服务对象的控制问题，而是一般地谈谈对于任何一种服务对象都要考虑的共同性问题。

虽然我们在前面已经指出对于6502可屏蔽中断服务程序入口，应该存放在规定的FFFE和 FFFF两个单元中。但是由于6502的软件中断指令BRK也要使用这一向量来指向其查询软件中断标志位B的专用服务程序，因此FFFE和FFFF两个单元总是已经为系统的监控程序所占用了，用户的中断服务程序入口并不能直接存放在FFFE和 FFFF两个单元中。而应该阅读该系统的技术手册，按其中所指出的地址来存放自己的中断服务程序入口。

例如ABPPLE Ⅱ就应该将中断服务程序入口地址存放在03FE和03FF两个单元之中(前者放低八位,后者放高八位); AIM-65则应放在A400和A401两个单元; SYM-1则是放在 A67E 和 A67F两个单元。

拿APPLE Ⅱ 来说，它的中断服务程序的一般模式应为:

```
IRQVECL   EQU   $03FE
IRQVECH   EQU   $03FF
            ⋮
          LDA   #>INTSERV ; 在初始化程序中送中断
                            服务程序入口到3FE和
          STA   IRQVECL     3FF两单元
          LDA   #<INTSERV
          STA   IRQVECH
            ⋮
            ⋮
INTSERV   PHA             ; 中断服务程序开始
          TXA             ; 保护现场，将A，X,Y
          PHA               各寄存器内容送堆栈保
          TYA               存
          PHA
            ⋮             ; 和请求中断的I/O 交换
            ⋮               数据
          PLA             ; 恢复现场，恢复Y、X、
          TAY               A各寄存器内容
          PLA
          TAX
          PLA
          RTI             ; 返回主程序
```

最后一条中断返回RTI 指令的功能在第一章中已作过叙述，为和中断响应周期中的操作相呼应，它首先要恢复标志寄存器P 的内容（这点是子程序返回指令RTS所没有的）然后恢复断点地址，于是程序就由服务程序返回到主程序被打断处,实现了中断返回。其它机器的中断服务程序也可以基本上照这个模式编写，这里就不再重复了。显然，如果你的中断服务程序并不影响 A,X，Y寄存器的内容，那么也不一定要把它们保护起来。此时可对上述模式稍加修改，但必须记住凡要在服务程序中将要影响其内容的寄存器，都要送堆栈保护，在返回之前再予以恢复。这点和子程序的写法要求是一致的。

以上我们对于中断的讨论都是针对一个中断源而言的。在一个稍复杂的系统中，往往会有多个中断源的中断请求同时出现，而且这些中断请求对其响应的紧迫程度也各不相同。也就是说有的中断请求必须立即响应，而有的中断请求可以稍晚点响应，我们就说他们的优先权不同。这个问题应如何具体解决呢？解决这个问题有两种办法，一是硬件的办法，一是软件的办法。两者都要首先对各中断源按其性质排好优先权大小。

若用硬件的办法来解决，可以使用优先权编码器将每个中断源的中断请求转换成相应的优先权代码。例如，采用74148优先权编码器和多路开关74LS157来进行此项工作。(如图2—31所示)。当有一个中断请求输入信号变低时，将使6502的$\overline{IRQ}$引线变低；同时74148的输出端将会有相应的三位二进制码出现。当同时有两个中断输入引线变低时,74148将按其优先权大的那个输入而决定其代码。在图2—31中74148的0～7个输入端，数字越大优先权也越大，它们所对应的编码由表2—3决定。

由图2—31所示，一共有八个中断源。通过74148的0～7端八根引线输入中断请求信号。每根引线在没有中断请求时，它们都是高电平，有中断请求时才变成低电平。例如若引线 4 所连接的

中断源有中断请求，它将使引线4变成低电平。由表可知输出端EO将变成1，C、B、A三个输出则为011。如果同时有两个输入引线同时变成低电平，由于在74148中是按数字越大优先权越大的原则编排的，所以这时输出端C、B、A的编码应由数字大的端子决定，而忽略数字小些的输入端也同时为0的情况。例如数字6和2的输入端同时为0，则C、B、A的编码仍可按表2—3可知为001。由

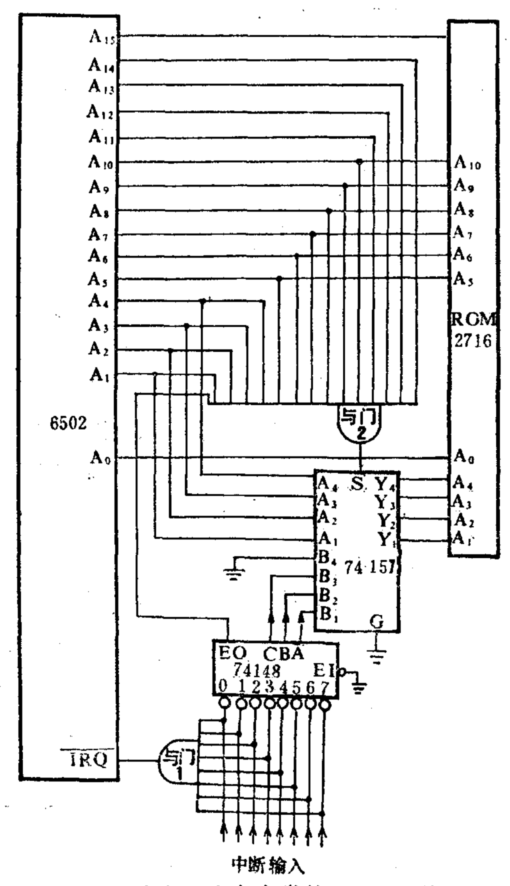


图2—31 多中断源中断向量的选择（硬件方法）

表2—3　　74148编码表

EI	输　入　端	C　B　A	GS　EO
1	7～0端为任何值	1　1　1	1　1
0	7～0端为全1	1　1　1	1　0
0	7端为0	0　0　0	0　1
0	6端为0	0　0　1	0　1
0	5端为0	0　1　0	0　1
0	4端为0	0　1　1	0　1
0	3端为0	1　0　0	0　1
0	2端为0	1　0　1	0　1
0	1端为0	1　1　0	0　1
0	0端为0	1　1　1	0　1

图2—31可知八根中断输入引线还同**与**门1相连，只要有一根引线变成低电平，**与**门1输出就使6502的$\overline{IRQ}$线变成低电平，而向6502提出中断请求。如果允许，6502将在完成正在执行的指令之后，即予以响应。在响应的最后过程是6502将从FFFE和FFFF中断向量中读取中断服务程序的入口地址送PC。在这两个读周期中，6502的地址线除A_0之外，A_1～A_{15}全部为高电平。由于这时EO也为1，因此这时**与**门2的输出将为高电平，而不是象在其它的情况下输出为低电平。由于**与**门2的输出端同数据多路开关74LS157的选择线S相连，因此我们参考表2—4所列出的74LS157的真值表即可知在这种情况下，它的输出不是由A_1～A_4所决定，而是由优先编码器74148的输出C、B、A所决定。到此我们可以看出6502虽然是固定地从 FFFE 和 FFFF 两个单元去读取中断服务程序入

口地址，但由于我们在硬件上进行了如图 2—31 的附加电路，而使得固定的一对单元转换成由八个不同的中断源所决定的八对相应不同的单元。我们可以在只读存贮器2716中八对相应的地址单元中事先写好相应的八个中断源所相应的服务程序的入口地址，因而就可以很圆满地解决多中断源的问题。

表2—4　74 LS157真值表

输入端				输出端
G	S	A	B	Y
1	×	×	×	0
0	0	0	×	0
0	0	1	×	1
0	1	×	0	0
0	1	×	1	1

采用硬件方法解决多中断源的问题的优点是响应时间很短，但明显的缺点是要增加好几个芯片。如果我们采用6500系列的可编程接口芯片（详见第四章）就不必这样做了。这些接口芯片内对每个中断源都已做好了一个中断标志位。当这个中断源要求申请中断时，将首先将它所对应的标志位置1，如果允许的话，同时就将使芯片的输出引线$\overline{IRQ}$由高电平变成低电平，而这根引线是与6502的输入引线$\overline{IRQ}$（注意两者同名但含义不完全相同）相连的。在这种情况下多中断源的问题一般就采用软件的方法来解决。

在多中断源情况下的中断服务程序首先应安排一个中断源查询程序，查询到中断源之后，再跳转到相应的服务程序。具体做法如下：

```
POLINT  PHA
        TXA
        PHA
        TYA
        PHA           ；保护现场
        LDA  IFR      ；读中断标志寄存器（IFR）的值
```

```
                    到累加器A
ASL  A      ; 累加器算术左移，C←bit7
BCS  ONE    ; 以下ONE、TWO、THREE都是
ASL  A        中断服务程序入口地址的标号
BCS  TWO
ASL  A
BCS  THREE
 ⋮
 ⋮
```

程序中使累加器A的值进行算术左移，再通过移入进位位C的值来决定是否转到该中断源的服务程序。程序中我们假定它们的入口分别是ONE、TWO、THREE……等。有关恢复现场后返回的指令应在各个中断服务程序中安排。由上面的程序可以看出，显然中断优先级别高的中断标志位在标志寄存器的高位。关于6502中断系统的具体应用在第四章及第五章还要再作具体介绍。

以下我们附带谈一个别的问题。这个问题虽然与中断无关，但它却与中断的响应过程有某些相似之处，这就是$\overline{RESET}$（复位）信号与复位过程。前面我们已经提到6502有一条$\overline{RESET}$输入引腿，如果外部将这条引线强迫成低电平，这就强迫6502停止工作。只有$\overline{RESET}$（复位）引线变成高电平以后，6502经过一个过程——我们称为复位过程——以后，才有可能使6502转入正常工作状态。当微型计算机电源接通以后，晶体振荡器立即起振，6502立即也得到正常的工作电源，但这时我们从电路上应使6502的$\overline{RESET}$（复位）引线处于低电平，经过很短一段时间（约几个毫秒）之后，这条引线的电位自动恢复成高电平。这时6502即开始了它的复位过程——即起动过程。这个过程需要延续6个时钟周期。在这期间6502自动从存贮器一对特殊指定的地址单元中取出二个八位二进制数，并将它们赋给6502的程序计数器PC。这

与中断响应的情况很类似。这一对特殊的地址单元所存放的数值一般就是系统软件中初始化程序的入口地址，所以我们常把这个值叫做起动向量。这一对特殊的地址规定是FFFC和 FFFD（十六进制）。在复位期间6502自动地首先去读存贮单元FFFC所存放的值,并将它送程序计数器 PC 的低八位,接下去6502又自动地去读FFFD（十六进制）存贮单元的值,并将它送程序计数器PC的高八位。从此之后，6502将开始正常工作，自然它将从起动向量所指的地址开始正常运行。 APPLE Ⅱ PLUS 微型计算机 在FFFC 和FFFD两存贮单元中存放的是62和FA。FA62正是 APPLE Ⅱ 监控程序的上电起动入口地址。（注意：APPLE Ⅱ PLUS 监控程序还有另一个版本。那里的上电起动入口地址是FF59。）AIM-65微型计算机的上电起动向量中存放的是EDBF（十六进制）。



第三章　以6502为CPU的微型计算机

本章拟对三种有代表意义的以6502为CPU的微型计算机先进行概括地介绍，然后再介绍它们的操作使用。这三种微型计算机的型号是SYM-1，AIM-65和APPLE Ⅱ PLUS。这三种计算机都由美国的公司所生产，但它们应用极为广泛，不仅在美国而且在全世界都有很多的用户。特别是APPLE计算机直到1981年一直在微型计算机市场占据首位，我国近年来也大量引进了APPLE机，广泛应用于文化教育、科研、国防、管理与控制等各个领域。SYM-1、AIM-65虽然都是单板计算机，但同Z80、8080/8085、6800的单板机相比，它们所拥有的硬件及软件资源明显地占着优势，在美国它们的应用也十分广泛。我国近年来对这两种单板计算机也引进了一部分，受到了用户很高的评价。但目前对它们还不熟悉，还没有充分认识它们的优越性，这也是我们在这里要对它们逐个予以介绍的原因之一。

在分别介绍它们之前，我们首先介绍一个最简单的以6502为CPU的微计算机结构。一个微计算机起码应包括：微处理机、时钟电路、ROM、RAM以及一个或多个PIO。这就是如图3-1所示的结构。对于6502来讲，时钟电路的大部分都已在芯片内部，因此只需外部接入一个石英晶体，6502时钟频率应为1兆赫。由图3-1可以看出，6502同其他微处理机一样，也有三条总线：即数据总线（8条线），地址总线（16条线）和控制总线。应当注意，8条数据总线均为双向，即数据信号在线上是可以两个方向传输，而地址总线的16条线都是单向的，即地址信号总是由6502经过地址总线向外送出的。由图可以看出RAM、ROM和PIO均同三总

线分别相连。ROM 中可以包含计算机的监控程序以及其它系统软件或者是用户的应用程序(如工业控制程序)。PIO 至少提供两个 8 位口和几条控制线同外部的设备相连。一个功能较完善的微计算机 PIO 芯片应有几片。RAM 是读写存贮器，它的容量大小往往决定了机器的功能，早年由于 RAM 价格较为昂贵，所以一般微计算机所具有的 RAM 不是很大，只有如1k字节、4k字节、16k字节、32k字节等等，近年来 RAM 的容量也越来越大。但是由于仅有十六条地址线，所以最大允许的存贮器(包括 RAM 和 ROM)的容量为64k字节，而且由于6502的输入输出采用存贮器映象方式，它没有单独的输入输出指令，PIO 等接口芯片也要同存贮器一道考虑统一编排地址空间，因此 RAM 和 ROM 所占的空间就只能小于64k字节了。

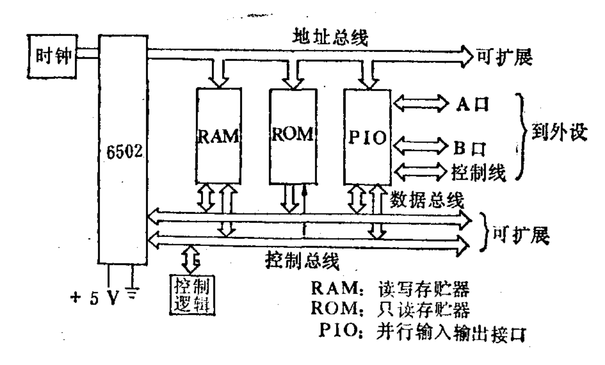


图3—1 简单的6502为CPU的微计算机结构

§3—1 SYM-1 单板计算机

图3—2表示出了 SYM-1 单板计算机方框图。由图可以看出单板计算机除MPU6502 之外，有4kB 只读存贮器，其中存放了它的监控程序 (SUPERMON MONITOR)；有 4kB 的读写存贮

器(RAM)；此外还有通用接口适配器(VIA)6522三片，支持着盒式录音机、示波器显示驱动等接口。其中一片6522完全为用户开发应用使用；还有一片RIOT(存贮器输入输出和定时器接口)6532芯片支持键盘、LED显示器以及CRT/TTY接口。6532中有128字节的读写存贮器，其中存放着监控程序所必须的数据，并且还有一部分提供作为缓冲区以及存放中断向量等。

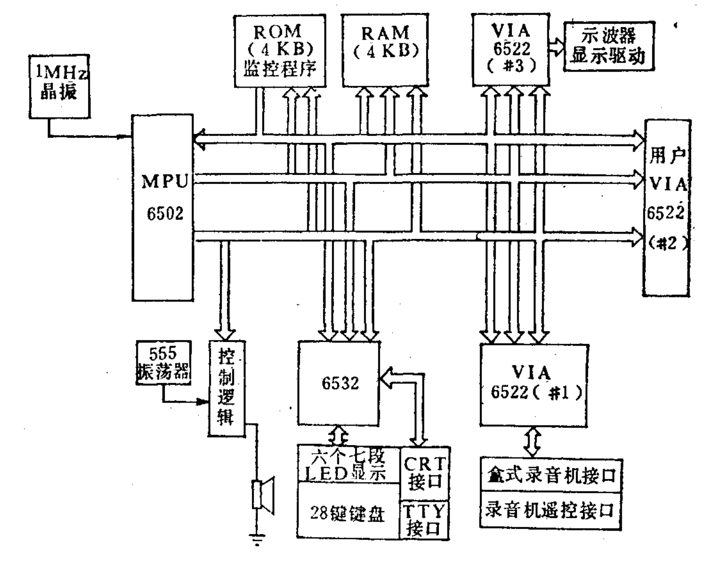


图3—2　SYM—1单板计算机方框图

表3—1列出了SYM—1存贮空间分配情况。SYM—1已经发展了自己的汇编程序和文本编辑程序以及8k BASIC语言解释程序，不过这些都是选择件，基本的SYM-1单板机是不具备的，而且当你选择它们时，还需要配上CRT终端或TTY（电传打字机）。目前引进的SYM-1都是一个基本的单板机，但其功能还是很好的。具体归纳如下：

（1）28键的键盘，每个键都是双功能键，因此较一般单板机

丰富。

(2) 磁带的存取可以按两种格式即KIM(每秒8字节)格式和SYM(每秒185字节)格式。而且每个存带的文件可以附加一个二位数的(100～FF)识别数字以作为文件名。在读带时可以按指定的识别数字对磁带进行搜索，一直到读进该文件为止。

(3) DEBUG功能很好，可以按任意指定的速率对程序进行

表3—1　SYM—1存贮空间分配

地址(十六进制)	用途
0000～00FF	00F8～00FF用于监控程序，其余用户可以使用。
0100～01FF	堆栈
0200～0FFF	用户RAM
1000～7FFF	不用
8000～8FFF	4kB超级监控程序ROM
8FFF～9FFF	备作监控程序的扩充
A000～A3FF	#1号VIA6522用
A400～A5FF	系统输入输出6522用
A600～A7FF	其中A600～A67F(128bytes)是由6532提供的系统RAM，其余未用
A800～ABFF	#2号VIA6522用
AC00～AFFF	#3号VIA6522用
B000～BFFF	备用。汇编/文本编辑程序ROM用。
C000～DFFF	备用。8kBASIC解释程序ROM用。
E000～EFFF	备用。汇编/文本编辑程序ROM用。
F000～FFFF	其中F800～FFFF为系统RAM的反射位置。

跟踪。

(4) 可以在一定的存贮区进行对某一指定内容的搜索；可以成块移动数据。

(5) 可以接一示波器做为字符显示器(一行)。

(6) 有八个可供用户定义的功能键。

一般单板机都有的功能我们就不一一枚举了。

§3—2 AIM-65单板计算机

AIM-65虽说是单板机，但它实际上具有的功能远远超过了一般的单板机，接近一个微型计算机系统。它有丰富的软件和较全的外设，所以有人说它是学习使用微型计算机最好的工具。它既可以用来学习高级语言(BASIC、FORTH、PL/1等)编写程序，也可以用来学习汇编语言实现各种控制功能。它象一般单板机那样便于与外界联系和扩充，又象一般系统机那样有丰富的软件和较齐全的外设。而它的价格却较低廉。所以这个机器在美国很受欢迎，国内引进以后也受到了很好的评价。

AIM-65具有两个盒式录音机接口，可以带两部录音机工作。TTY(电传机)接口、20列热敏打印机接口及20列热敏打印机，字符显示器接口及20列16段 LED 字符显示器。此外 AIM-65还有一个54键的全功能键盘及接口，它除了8k监控程序、汇编及文本编辑程序和 BASIC 解释程序等软件之外，还有 PL/1，FORTH等高级语言可供选择。它专有一片通用接口适配器 VIA 6522供用户进行开发应用，并设有专门的应用总线，与外界联系很方便。它的 RAM 只有4kB，似乎有些小，但它设有扩展总线可以方便地将内存扩展。

AIM-65的监控程序共8k，功能很强。磁带、打印机、显示器等的管理程序以及反汇编和小汇编程序等等都包括在内。AIM

-65磁带上的文件可以按文件名进行存取；可以用 KIM，AIM 两种格式读写磁带。打印机可用来打印编辑的文件，汇编的清单，存贮单元内容、程序以及计算结果等等一系列的很有用的清单。字符显示器是由20个16段发光二极管(LED)和显示寄存器组成。由于有这些硬件因此处理显示问题较一般单板机容易，占用CPU时间少了许多。显示器虽只有一行20个字符，不如 CRT 显示好，但由于有打印机可印出清单，可以弥补显示器的不足。监控程序中还包括了初等汇编和反汇编程序，这给调试程序带来了很大的方便。一般几十条指令的小程序可直接用初等汇编输入和修改，反汇编程序可直接将存贮器中的机器码形式的程序列出以记忆符表示的程序清单。监控程序中的跟踪功能，可以让打印机快速打印出程序执行过程中程序计数器、累加器等的值，这对于程序的调试排错很有用。

AIM-65 主板上有 PROM 的三个空插座，原设计是插入三片 TEXAS 2532 (EPROM 4k×8)，略加修改仍可改插三片 INTEL 2732 (EPROM 4k×8)。一般情况下这三片应为：一片为汇编/文本编辑程序，另二片为 8kB BASIC 语言解释程序。这些固化了的软件其功能也是很强的，ROCKWELL 公司已有出售。它的 BASIC 语言除了没有绘图部分以外其余与 APPLE Ⅱ 机的 APPLESOFT BASIC 语言几乎完全相同。汇编程序和文本编辑与 APPLE Ⅱ 机也很相似。当然在 8k BASIC 插座的位置上也可改插固化了的 FORTH 和 PL/1 等高级语言的编译程序。

图3-3是 AIM-65单板计算机的方框图。图中未画出电源。AIM-65配有一个专门的电源，其中包括稳压的+5V(2A)和不稳压的+24V(2.5A)。前者供主机使用，后者供打印机用。由图可以看出 AIM-65中除6502之外，主要是还使用了6500系列的几个可编程接口芯片。其中包括二片6522(VIA)，一片6532(RIOT)和一片6520(PIO)。它们的功能和用法下一章将详细讨论。其中有一

表3—2　　AIM-65 存贮空间分配

起始地址	终止地址	内容
0000	00FF	0页—系统RAM。部分单元用户可用
0100	01FF	系统RAM以及堆栈
0200	0FFF	用　户　区
1000	9FFF	用户可利用的扩充地址
A000	A00F	用户6522(VIA)
A010	A3FF	不使用
A400	A47F	监控程序RAM(由6532提供)
A480	A497	6532(RIOT)　键盘I/O
A498	A7FF	不使用
A800	A80F	6522(VIA)打印机，电传机和盒式录音机
A810	ABFF	不使用
AC00	AC03	6520　显示器I/O
AC04	AFFF	不使用
B000	CFFF	BASIC　ROM
D000	DFFF	汇编程序/文本编辑程序　ROM
E000	FFFF	监控程序ROM

片6522通过应用总线专供用户进行实时控制方面的开发应用。

表3—2列出了 AIM-65 的存贮空间分配。

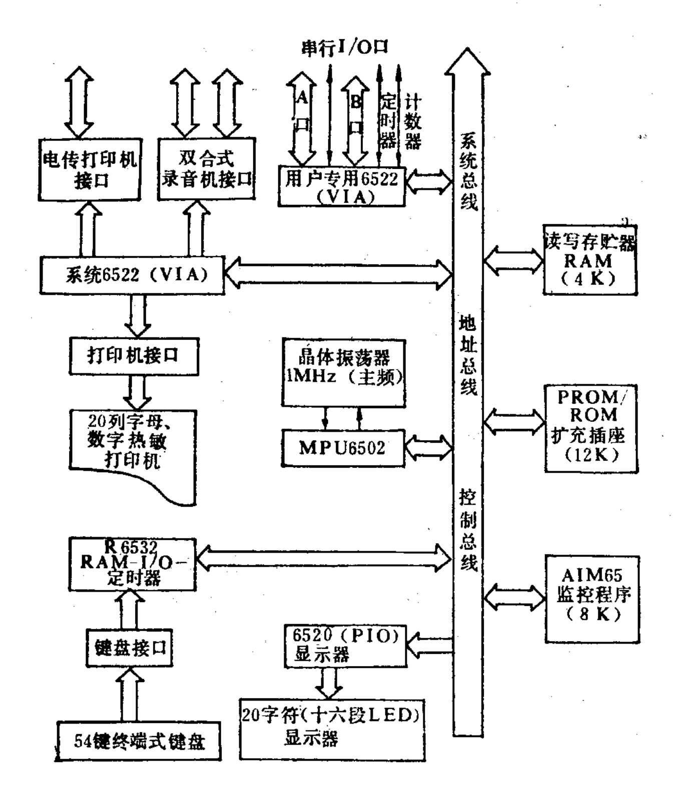


图3—3 AIM-65 单板计算机方框图

§3—3 APPLE Ⅱ PLUS微型计算机

前面已经提到过 APPLE Ⅱ 机是举世闻名的，自1977年问世以来已经在全世界销售了一百万台以上，它拥有丰富的软件以及各种配件和附件是任何一种别的微型机可以与之相匹敌的。我

国近年来大量引进了APPLE机，并已经把它列入典型系列机的行列，在教育部门特别是在中小学它最受广大师生的欢迎，本节拟对它进行概括地介绍。

首先我们来介绍APPLE Ⅱ 主机的系统结构。图3-4画出了它的方框图。由图可以看出它是以6502为中央处理机的，同别的微机一样6502有三条总线：即数据总线、地址总线和控制总线同外界联系。APPLE机有12kB的只读存贮器，这里存放着系统软件：监控程序和BASIC解释程序。如果BASIC解释程序选择的是APPLESOFT浮点BASIC语言，那这样的APPLE机称为APPLE Ⅱ PLUS；如果选择的是整数BASIC语言，那这时的APPLE机称为APPLE Ⅱ。前者是后者的改进型，于1979年推出。在此之前生产的是APPLE Ⅱ 机，近年来已不再生产APPLE Ⅱ，而改为生产APPLE Ⅱ PLUS。APPLE Ⅱ PLUS所拥有的读写存贮器RAM规定为48kB，而不象早期的APPLE Ⅱ，它的RAM有16kB、24kB、32kB…等。1983年APPLE公司为防止伪造，推出APPLE Ⅱe。其实Ⅱe与Ⅱ PLUS软件完全兼容，硬件则将RAM扩大为64kB，以及用大规模集成电路的几个专用芯片代替了原来用中小规模集成电路芯片做成的板上接口以及电视信号的有关部分等。目前国内绝大部分仍为APPLE Ⅱ PLUS机。

APPLE机内的主板上有八个空的插槽，它们是供用户选择各种不同的外设接口板，以根据用户的需要组成其系统。每个槽口有50条线——这就是APPLE总线。按照APPLE总线的要求可以对APPLE机进行各种开发应用。APPLE机主机板上已经给用户提供了几种接口，这就是我们在图3-5上画出的录音机接口、键盘接口、喇叭接口以及游戏控制器接口。有了录音机接口使用户可以用一个普通的盒式录音机做外存贮器，而不必一定要购买较昂贵的软盘驱动器。APPLE机的主机上已包括了键盘接口和键盘。原来APPLE机的键盘只有52个键，近年来国产的仿APPLE

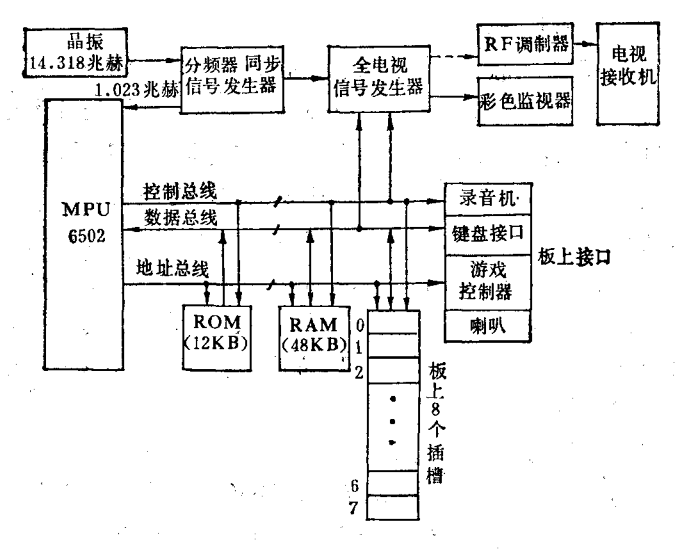


图3—4 APPLE 计算机框图

Ⅱ机都改进了这方面的功能，加上了大小字母控制键，有的甚至还按汉字系统的需要在每个键的侧面刻上了汉字的基本字根和偏旁。APPLE 机内装有一个 2 吋小喇叭，在游戏和辅助教学的程序中可以用它来提醒操作者。APPLE机的游戏控制器接口包括四个摸拟量输入、三个开关量输入和四个开关量输出。它们既可用来做玩游戏的操纵杆，也可以用来做某些控制实验的输入输出开关量和模拟量。

APPLE 机可分美国型的和欧洲型的两种，现在国内流行的多为美国型。所谓美国型与欧洲型主要是它们的晶振以及同步信号和彩色全电视信号不相同。美国型的 APPLE 机产生的是NTSC制式彩色全电视信号，在这种情况下，晶振频率为14.318兆赫，经四倍分频后产生3.58兆赫(近似)的彩色付载频。欧洲型的APPLE 机晶振频率为17.73450兆赫，经四分频后产生4.43兆赫的

彩色付载频，前者应配NTSC制的彩色监视器，而后者则应配PAL制彩色监视器。

APPLE机的软件是极丰富的，不过绝大多数软件的工作环境需要在图3—5所示的八个插槽中加上两块板。一块是软磁盘驱动器控制板，插在槽号6；另一块是扩展16kB RAM板，插在0号槽口上。当然为了工作方便还往往要再加上打印机接口板，插在1号槽口上。APPLE机的典型配制是主机（包括16kB扩展RAM板）、显示器、再加上两个软盘驱动器（包括接口），以及打印机和接口；而简化的配制则是主机、录音机、显示器。这种配置只能在其上运行APPLESOFT BASIC程序，但在这种机器上运行的软件依然可以在典型配置的机器上运行。

以下我们对典型配置的APPLE机谈谈它的软件。APPLE公司为它的机器提供了两种操作系统：即DOS3·3和UCSD-PASCAL操作系统。前者支持APPLESOFT扩展浮点BASIC语言，整数BASIC (INTEGER BASIC)，LOGO语言、汇编/文本编辑程序以及一大批应用程序，如PLOT及各种游戏程序等。UCSD-PASCAL操作系统支持PASCAL语言，APPLE FORTRAN语言，PILOT语言以及一大批应用程序。为了扩大APPLE机的用途，并非APPLE公司而是另一个公司在其上开发了一块名称叫SOFTCARD的插件板，该板使APPLE机可以既工作在6502又工作在Z80的控制之下，而成为一个双CPU（中央处理机）的微型计算机。同时在这块板的支持下，可以运行CP/M操作系统。在CP/M支持下，可以运行一大批应用程序，可以在APPLE机上对用8080的指令写的源程序进行汇编。为了在APPLE机上可以使用汉字，不少厂家在其上开发了汉字系统。其中使用比较多的是台湾的佳佳汉卡（或仓颉汉卡）。该系统采用中文字母法输入汉字，字库有两万三千余汉字，价格较一般汉字系统便宜得多，缺点是该系统是繁体字，而且只能在APPLESOFT BASIC语言下

调用。APPLE机应用软件极为丰富，据统计有一万六千个以上，比较著名的如 VISICALC—专门进行数据表格处理方面的软件，特别适合财会、工资、学员成绩等方面应用。还有 WORDSTAR 应用软件,特别适用于文字处理。此外 APPLE机的附件与配件也较别的机种丰富得多,如 D/A和A/D转换板，ROM写入板，RS232接口，IEE488接口，80列卡等都应有尽有。外设除了上述的典型配置以外，还可以配绘图板输入装置，绘图仪输出装置，外存还可以配5MB-30MB温氏硬盘。近年来发展的计算机局部网，如 OMNINET NESTAR 以及 APPLE 公司自己发展的局部网都可以使用 APPLE Ⅱ PLUS 机。如果说一台APPLE机的功能是有限的，那么联成局部网以后将使其功能大大加强，而价格却较完成同样功能的小型计算机系统要便宜一半左右。

以下我们谈谈 APPLE 机 的内存分配情况。由表3—3可以看出RAM的 0 页，1页和 2 页（即0000～02FF）都和系统密切相关。其中 2 页（0200～02FF）共256个单元是用作为键盘的一个输入行的缓冲区，即当你从键盘打入程序、命令等时，在你没有按回车键之前，这些字符都以 ASCII 码的形式存放在这个缓冲区内。由于它的容量是256个单元，因此 APPLE 机的任何一个程序行所包含的字符的个数一定要小于256。RAM 的第 1 页（0000～01FF）是系统的堆栈。这是由6502微处理机所决定的。由于这一页只有256个单元，因此子程序的嵌套层数受到它的限制，不过很少有程序会达到这个极限。6502的指令系统中采用零页寻址方式的指令很多，因此零页（0000～00FF）在系统软件中占有非常重要的地位，整个零页256个单元除了为数很少的十余个单元未用以外，其余绝大部分为系统所占用。以下我们列出一些常用的地址单元：

20（十六进制）：文本窗口左边框位置单元。APPLE机把屏幕从左到右分为40列，列号为0～39。第 0 列为屏幕的最左边一列；而第39列则为屏幕的最右边一列。如果在20单元内存放的值为 0

表3—3　　APPLE机内存空间分配

起始地址	结束地址	内容
0000	00FF	系统RAM
0100	01FF	系统堆栈
0200	02FF	输入缓冲区
0300	03FF	部分用于监控程序
0400	07FF	文本/低分辨度图形第一页
0800	0BFF	文本/低分辨度图形第二页
0C00	1FFF	用户区
2000	3FFF	高分辨度图形第一页
4000	5FFF	高分辨度图形第二页
6000	BFFF	用户区
C000	CFFF	输入/输出
D000	FFFF	BASIC解释程序和监控程序

则所显示的文本由屏幕最左边开始；若改变20单元内存放的值为5，则显示的文本将从屏幕的第5列的位置开始。注意，20单元值变化并不能改变文本窗口宽度，只是移动了边框的位置。但是如果20单元的值大于39或它的值加上文本窗口宽度的值大于40，则输出的文字资料将全部或部分不被显示在屏幕上，这将使你损失程序或数据的某些部分。

21(十六进制)：文本窗口宽度单元。这个单元的值将确定屏幕上所显示的文本的宽度，它的值必须在1到40之间选择，它的值总是等于屏幕上右边框与左边框相距的列数。注意不要在21单元内存放0，否则将使APPLE机的BASIC解释程序不能正常工作。

在启动APPLE机之后，监控程序将会在20(十六进制)单元内存放0，21(十六进制)单元内放28(十六进制，即十进制40)。

22(十六进制)：文本窗口的上边框单元。这个单元的值将决定屏幕上所显示的文本的上边框位置。APPLE机把整个屏幕分成24行，行号是0～23。第0行的位置在屏幕的顶端，而第23行则在屏幕的底端。一般22(十六进制)单元内存放的数值是0，所以显示在屏幕上的文本总是从顶端开始。

23(十六进制)：文本窗口的下边框单元。这个单元存放的数值的大小确定着在屏幕上所显示的文本的下边框的位置。它的值应在0～23之间选取，但要注意千万不要将前面介绍的上边框单元和这里的下边框单元的值弄颠倒了，总是要求上边的值应小于下边框的值。

24(十六进制)：游标水平位置单元。这个单元内存放的数值确定着当前游标在屏幕上的水平位置，它的值正好等于游标与文本窗口左边框之间相距的列数。如上所述屏幕分成0～39列，因此游标的水平位置可以是0～39之间的任何一个值。

25(十六进制)：游标的垂直位置单元。这个单元内存放的值决定着游标当前的垂直位置，它的值正好等于游标与屏幕顶端相距的行数。如上所述，我們已将屏幕分成0～23行，因此它的值应为0～23。

30(十六进制)：低分辨度图形色彩单元。APPLE机有两种图形显示方式，一种为低分辨度，另一种为高分辨度。前者可以显示十六种不同的色彩，这些色彩由放在30(十六进制)单元的值

所决定。(如表3—4)由于30(十六进制)单元有八个二进制位，相当两个十六进制数，因此表3-4所对应的数值应重复写两遍。例如粉红色，应在30单元中放十六进制数BB；紫色则应放十六进制数33。依此类推。

表3—4　　　　　　　　颜　色　代　码

数值(十六进制)	颜　色	数值(十六进制)	颜　色
0	黑	8	棕色
1	洋红	9	桔黄
2	深蓝	A	灰色2
3	紫色	B	粉红
4	深绿	C	浅绿
5	灰色1	D	黄色
6	蓝色	E	海蓝色
7	浅蓝	F	蓝色
			白

32(十六进制)：决定字符显示方式的单元。APPLE 机字符显示有三种方式：即正常方式、反相方式和闪烁方式。所谓正常方式即背景为黑色，字符为白色；反相方式系指背景为白色，字符为黑色。正好同前者相反，故称作反相方式。产生这样三种不同的方式，是由存放在32单元的值来决定的。

如果(32)＝FF(十六进制)，则字符为正常方式显示；如果(32)＝3F(十六进制)，字符显示为反相方式；若(32)＝7F(十六进制)，则显示的字符将不停地闪烁。

现在我们再回过头来分析表3-3。APPLE 机RAM 的第 3 页(0300～03FF)有一部分将是一些很重要的向量，另一部分则可以

是用户程序区。例如3F2和3F3(十六进制)两个单元将存放着软件入口地址，这是正在使用的高级编译程序语言的入口地址。如一般情况下它们存放着E003(十六进制)，即(3F2)=03，(3F3)=E0。另外3FE和3FF单元是中断向量，这一点已经在上一章讲述。

由表3-3中可以看出分配了0400～07FF作为文本/低分辨图形第一页的存贮区，而0800～0BFF则作为文本/低分辨图形第二页的存贮区。APPLE机的特点是用电视监视器作为输出显示，因此显示所需的刷新存贮器只能由计算机提供，所以APPLE机将内存RAM的一部分用作显示的刷新贮存器。如表3-3所示，地址2000～3FFF以及4000～5FFF也都用作了高分辨度图形显示的刷新存贮器。这里无论是低分辨度或是高分辨度图形都安排两页是为了用户在显示图形时可以将它们用来进行动画显示，但这样无疑将大大地减少用户的存贮空间，不过往往在APPLE机用于管理时，高分辨度图形一般并不使用，那这种情况下，上述的2000～5FFF(16kB)依然可以用来存放程序和数据。APPLE机低分辨度图形显示时将屏幕分为40×48个方块，每个块可以在表3-4所列出的十六种颜色中任选一种，因此它可以显示出美丽的色彩斑烂的图形。为了方便APPLE机，还安排了一种混合方式显示，即40×40个方块的低分辨度图形，在图形的底部安排了四行文本，可以显示字符。而高分辨度图形则将屏幕分成280×192个点，可以在黑、白、蓝、绿、桔、黄、紫六种颜色中任选。高分辨度图形显示时也可以有混合方式，即屏幕划分为280×160个点组成，图形下方有四行文本。这些屏幕显示方式如何控制呢？无疑在一个时刻，荧光屏只能按一种方式显示计算机的输出。那么这些显示方式之间是如何实现相互转换的呢？在启动电源之后，监控程序将使屏幕设置成文本显示方式，即自动将0400～07FF这些单元用作为屏幕显示字符的刷新存贮器。每在屏幕上显示一个字符就必须在这个存贮区所对应的单元存放其相应的ASCII码。例如要在荧

屏左上角显示一个字母"A"， 已 知 左上角所对应的存贮单元是0400(十六进制) 那么就要将 "A" 的ASCII码C1 (十六进制) 放进0400(十六进制)单元中去。如果我们要将文本显示方式转换成别的什么方式， 就需要使用APPLE机所专门设计的几个屏幕 软 开关(如表3-5)。所谓软开关是指它们确实起着屏幕显示方式的转换

表3—5　　APPLE 屏幕软开关

APPLE 屏幕软开关		
地（十六进制）	址（十进制）	功　　能
C050	49232－16304	图形显示方式
C051	49233－16303	文本显示方式
C052	49234－16302	全屏幕显示文本或图形
C053	49235－16301	文本图形混合方式
C054	49236－16300	显示第一页
C055	49237－16299	显示第二页
C056	49238－16298	低分辨度图形方式
C057	49239－16297	高分辨度图形方式

作用，就象开关一样。"软"是说这些开关是采用编程软件控制的方式，而不是象普通开关那样用机械力或电信号控制。控制这些开关的办法是用6502的读或写指令对该地址进行读写。例如：

```
LDA  $C055
LDA  $C052
LDA  $C057
LDA  $C050
```

将使APPLE机处于全屏幕第二页高分辨度图形显示方式。若要转

成全屏幕第一页文本方式显示则应写：

```
LDA    $C054
LDA    $C051
```

依此类推，可以进行各种显示方式之间的转换。表3-5中列出了地址的两种十进制数表示方式。后者是将前者以65536为模而求出的补数，即将65536减去地址数的正数再变号，就得了这个补数。

表3-3中从6000起到BFFF为用户区，不过在有软磁盘驱动器的系统，这个区域中还将要存放磁盘操作系统。如使用DOS3·3操作系统时，存贮器9600～BFFF将用来存放操作系统以及文件输入输出缓冲区。

由于6502的输入输出是采用存贮器映象方式，也就是说要在存贮空间中开辟一部分空间作为输入输出，因此APPLE机专门在存贮空间中开辟C000～CFFF作为输入输出(见表3-3)。所有的板上接口以及将要在八个空插槽加入的接口，都要安排在这4kB的空间内。以下我们首先列出板上接口的地址分配情况：

功能	地址（十六进制）	地址（十进制）		操作
键盘数据	C000	49152	-16384	读
清除键盘打入脉冲	C010	49168	-16368	写
喇叭	C030	49200	-16336	读
盒式录音机输出	C020	49184	-16352	读
盒式录音机输入	C060	49256	-16288	读
游戏控制器：				
电位器(一)	C064	49252	-16284	读
(二)	C065	49253	-16283	读
(三)	C066	49254	-16282	读
(四)	C067	49255	-16281	读
清除电位器输入	C070	49264	-16272	读/写

开关量输出 0	关	C058	49240	-16296	写
	开	C059	49241	-16295	
1	关	C05A	49242	-16294	写
	开	C05B	49243	-16293	
2	关	C05C	49244	-16292	写
	开	C05D	49245	-16291	
3	关	C05E	49246	-16290	写
	开	C05F	49247	-16289	
开关量输入 0		C061	49249	-16287	读
1		C062	49250	-16286	读
2		C063	49251	-16285	读
通用打入脉冲		C040	49216	-16320	读/写

以上这些就是板上接口所分配的地址，对这些地址进行上面所指出的读或写操作就可以完成你所要求的任务。

前面已经提到过APPLE机上有八个空的插口或外设连接器，这八个插口列为0～7号。其中0号是特殊的，它专为扩展RAM到64k之用，而1～7号则是外设接口插槽，你可以任意插入那些专为APPLE机设计的接口板。为了使其比较简单和更有通用性，APPLE机内部在这七个插口的每一个上安排了总数达280个地址单元的存贮空间，另外还有2kB的通用空间，所有的接口板可以分时使用。分配如下：

1 号槽口　C100～C1FF　(十六进制)

2 号槽口　C200～C2FF　(十六进制)

3 号槽口　C300～C3FF　(十六进制)

4 号槽口　C400～C4FF　(十六进制)

5 号槽口　C500～C5FF　(十六进制)

6 号槽口　C600～C6FF　(十六进制)

7 号槽口　C700～C7FF　(十六进制)

这对于每个槽口(1～7号)是256个存贮地址单元,它们一般用来安排每个接口板上的只读存贮器，以存放引导程序或管理外设的子程序。如果需要大的子程序或管理程序可以在接口板上安排一个只读存贮器地址是C800～CFFF(十六进制)。2kB空间的只读存贮器在每个接口板上都可以安排一块，但只有启动该板的外设时，才能使它“使能”(ENABLE)，而其它板的只读存贮器都处于禁止状态,这样就实现了分时共享C800～CFFF(十六进制)这 2kB存贮空间的目的。

此外，每个槽口(0～7号)还安排了另外十六个通用输入输出存贮单元，分配情况如下：

0 号槽口	C080～C08F	(十六进制)
1 号槽口	C090～C09F	(十六进制)
2 号槽口	C0A0～C0AF	(十六进制)
3 号槽口	C0B0～C0BF	(十六进制)
4 号槽口	C0C0～C0CF	(十六进制)
5 号槽口	C0D0～C0DF	(十六进制)
6 号槽口	C0E0～C0EF	(十六进制)
7 号槽口	C0F0～C0FF	(十六进制)

除了上述以外,APPLE机还在其RAM中巧妙地安排出64个单元——每个槽口8个单元，以作为每个接口板的高速暂存器。它们的地址分配如下：(十六进制表示)

0号	1号	2号	3号	4号	5号	6号	7号
0478	0479	047A	047B	047C	047D	047E	047F
04F8	04F9	04FA	04FB	04FC	04FD	04FE	04FF
0578	0579	057A	057B	057C	057D	057E	057F
05F8	05F9	05FA	05FB	05FC	05FD	05FE	05FF
0678	0679	067A	067B	067C	067D	067E	067F
06F8	06F9	06FA	06FB	06FC	06FD	06FE	06FF
0778	0779	077A	077B	077C	077D	077E	077F
07F8	07F9	07FA	07FB	07FC	07FD	07FE	07FF

注意：上面的表中所列出的地址虽然都是处在文本/低分辨度图形第一页的存贮区，但由于这个区域共有1024个存贮单元，而APPLE机显示24行文本，每行40个字符只一共用去960个存贮单元，尚余下64个单元未曾使用。这正是上面所列出的那64个存贮单元。

综上所述，每个槽口除可以分时享用C800～C8FF(十六进制)2kB存贮空间外，还可以单独分别占有280个存贮地址单元。

我们再回到表3-3。从D000～FFFF(十六进制)是供APPLE机存放系统软件的存贮空间。APPLE Ⅱ PLUS机的主印刷板上有六片2kB的只读存贮器，存放着监控程序和APPLESOFT的浮点BASIC解释程序。当APPLE在使用别的高级语言时，其编译程序仍然放在这个空间，不过这时主板上的只读存贮器已经被“禁止”了。

§3—4 APPLE Ⅱ 监控程序 小汇编程序及编辑/汇编程序的使用方法

一. 监控程序命令

由BASIC转入监控(MONITOR)的操作为:

]CALL-151 ‹CR›

*

注: ‹CR›表示按RETURN键。划线部分是要求使用者按键打入的内容，不划线部分是APPLE Ⅱ自动显示出的内容，以下同。

当监控提示符*显示之后就可以使用以下的监控命令。

1. 检查存贮器的内容

(1) 检查单个内存地址内容

操作举例

```
*E000  ‹CR›        检查E000单元的内容
E000-   20
*300    ‹CR›
0300-   99
```

(2) 检查多个内存地址的内容

操作举例

```
a) *300    ‹CR›        检查0300单元的内容
0300-   99
*.315   ‹CR›           检查从当前地址0301单元
                       至0315单元的内容
0301-   00   08   C8   D0   F4   A6   7B
0308-   89   00   08   0A   0A   0A   99   A9
```

0310 - 09 85 27 AD CC 03

* .320 <CR> 检查从当前地址0316单元至0320单元的内容

0316 - 85 41

0318 - 84 40 8A 4A 4A 4A 09 4A

0320 - C0

*

b) * 300. 320 <CR> 检查从0300单元至0320单元的内容

0300 - 99 00 08 C8 D0 F4 A6 2B

0308 - 89 00 08 0A 0A 0A 99 A9

0310 - 09 85 27 AD CC 03 85 41

0318 - 84 40 8A 4A 4A 4A 09 4A

0320 - C0

*

c) * 300 <CR> 检查0300单元的内容

0300 - 99

* <CR> 继续检查包括0300单元在内的连续8个单元内容

00 08 C8 D0 F4 A6 2B

* <CR> 继续检查连续的8个单元内容

0308 - 89 00 08 0A 0A 0A 99 A9

* <CR> 继续检查连续的8个单元内容

0310 - 09 85 27 AD CC 03 85 41

*

2. 改变内存单元的内容

操作举例

a) *06 <CR>　检查06号单元的内容

0006-　00

*：5F <CR>　将06号单元的内容改为5F

*06 <CR>　再检查06号单元的内容

0006-　5F

*

如果要将某个单元的内容置为某值，可以简单操作为：

*302：42 <CR>　将302单元内容改为42

*302 <CR>

0302-　42

*

b) *300：69　01　20　ED　FD　4C　00　03　<CR>

从300号单元开始连续修改8个单元

*300 <CR>

0300-　69

*　<CR>

01　20　ED　FD　4C　00　03　连续的8个单元内容已改

*0310：0　1　2　3 <CR>　修改0310-0313单元

*：　4　5　6　7 <CR>　修改0314-0317单元

*310 · 317 <CR>　检查310-317单元

0310-　00　01　02　03　04　05　06　07

*

3. 移动一段内存单元的内容

命令格式：

{新位置的首地址}<{原位置首地址}·{原位置末地址}M

注：格式最后用“M”表示MOVE命令

操作举例：

```
*6000.600F ‹CR›        先查看6000-600F单元的内容
6000-   5F  00  05  07  00  00  00  00
6008-   00  00  00  00  00  00  00  00
*300: A9  8D  20  ED  FD  A9  45  20
DA  FD  4C  00  03 ‹CR›  修改300-30C单元
*300·30C ‹CR›          查300-30C单元置进的内容
0300-   A9  8D  20  ED  FD  A9  45  20
0308-   DA  FD  4C  00  03
*6000<300·30CM ‹CR›    将300-30C单元的内容
                       移到首址为6000的地方
*6000·600C ‹CR›
6000-   A9  8D  20  ED  FD  A9  45  20
6008-   DA  FD  4C  00  03    已移动
*310<6008·600AM ‹CR›   将6008-600A单元内容
                       移到310-312单元
*310·312 ‹CR›
0310-   DA  FD  4C
```

4. 比较两段内存单元的内容

命令格式：

{第二段的首地址}<{第一段的首地址}·
{第一段的末地址}V

注：格式最后用"V"表示VERIFY命令

操作举例

```
*6000: 0 1 2 3 4 5 6 7 8 9 A B C ‹CR›
*300<6000·600CM ‹CR›   将6000-600C单元内容
                       移到300-30C单元
```

```
*300<6000·600C V ‹CR›    比较这两段地址的内容
                         是否一致
*                        比较结果一致
*6006:66 ‹CR›            将6006号单元内容06修改为66
*300<6000·600C V ‹CR›    再比较这两段地址的内容
*6006-  66  (6)          比较结果6006号单元内容不一致
```

注：比较结果一致接着出提示符*。比较结果不一致就显示第一段地址中不一致的单元号及其内容并在括号内显示第二段对应地址单元中的内容。

5. 输入与运行机器语言程序

(1) GO命令

命令格式

{首地址}G

例 输入和运行一个显示字母A-Z的机器语言程序

根据题目要求写成下列程序(为了阅读方便写出助记符)

```
300  A9  C1         (LDA  #$C1)
302  20  ED  FD     (JSR  $FDED)
305  18             (CLC)
306  69  01         (ADC  #$01)
308  C9  DB         (CMP  #$DB)
30A  D0  F6         (BNE  $0302)
30C  60             (RTS)
```

说明：当执行G命令时，6502微处理机从首地址开始执行机器语言程序。监控程序把它当作一个子程序，当执行完后，只需执行一条RTS(从子程序返回)指令，控制就会回到监控程序，所以此程序最后一条是RTS。

操作示范：

```
*300: A9  C1  20  ED  FD  18  69  01
      C9  DB  D0  F6  60 <CR>   输入程序的机器码
*300.30C <CR>
0300-  A9  C1  20  ED  FD  18  69  01
0308-  C9  DB  D0  F6  60
*300G <CR>
ABCDEFGHIJKLMNOPQRSTUVWXYZ   运行结果
*
```

(2) LIST命令

命令格式

{首地址}L

执行LIST命令的结果，从指定的首地址开始，显示满屏幕的指令(20行)

操作举例：

```
*300L <CR>
0300-  A9  C1          LDA  #$C1
0302-  20  ED  FD      JSR  $FDED
0305-  18              CLC
0306-  69  01          ADC  #$01
0308-  C9  DB          CMP  #$DB
030A-  D0  F6          BNE  $0302
030C-  60              RTS
030D-  00              BRK
030E-  00              BRK
030F-  00              BRK
0310-  00              BRK
0311-  00              BRK
```

```
0312-   00              BRK
0313-   00              BRK
0314-   00              BRK
0315-   00              BRK
0316-   00              BRK
0317-   00              BRK
0318-   00              BRK
0319-   00              BRK
*
```

6．检查和改变寄存器的内容

EXAMINE命令的格式采用 CTRL E 键命令

操作举例：

```
* CTRL E  <CR>
A=0A  X=FF  Y=D8  P=B0  S=F8
(显示当时寄存器的值)
* :B0  02  D8  C0  F8 <CR>    修改寄存器内容
* CTRL E  <CR>
A=B0  X=02  Y=D8  P=C0  S=F8
*                              (显示改后内容)
```

注：若要改变后面寄存器的内容，前面寄存器内容要重写。例如要修改P的内容，前面A、X、Y重写原有内容

```
* : B0  02  D8  32 <CR>
*
```

7．其它几种监控命令

(1) 改变显示方式

正常显示方式为黑底白字，监控程序使用INVERSE命令(*

后打一个字母I ‹CR›）就改变为反显示方式（白底黑字），再使用NORMAL命令（*后打一个字母N‹CR›）回到正常显示方式

操作举例：

*0：10 11 12 13 14 15 16 17 ‹CR›

*0·7 ‹CR›

0000- 10 11 12 13 14 15 16 17

* I ‹CR› 反显示方式

* 0·7 ‹CR›

0000- 10 11 12 13 14 15 16 17

* N ‹CR› 正常显示方式

* 0·7 ‹CR›

0000- 10 11 12 13 14 15 16 17

*

（2） PRINTER命令

格式

{插座号} CTRL P

插座号可以是1—7

如果想将信息送到打印机去打印，它的接口插座在1号插座上，打*1 CTRL P ‹CR› 命令便可打开打印机，以后屏幕的所有显示便打印出来。

打*0 CTRL P ‹CR› 便停止打印机，回到原来的屏幕显示方式。

对于APPLE Ⅱe，此命令应改成PR#1和PR#0的格式

（3） CTRL Y 命令

使用此命令，可以强迫监控程序跳到＄3F8的位置，如果将转

移指令放在＄3F8单元，那就可以跳到所需的程序。

操作举例：

```
*3F8:4C  00  03 <CR> (3F8起存放JMP$300)
*CTRL Y    <CR>
ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

(4) 十六进制加减法

格式：

{数值}+{数值}

{数值}-{数值}

操作举例

```
*20+13 <CR>
=33
*4A-C <CR>
=3E
*3-4 <CR>
=FF
*
```

(5) 多重命令

多重的命令可以放在同一行上

操作举例

```
a) *300L 300G <CR>
   0300-  A9  C1           LDA  #$C1
   0302-  20  ED  FD       JSR  $FDED
   0305-  18               CLC
   0306-  69  01           ADC  #$01
   0308-  C9  DB           CMP  #$DB
```

```
030A-   D0  F6              BNE   $0302
030C-   60                  RTS
030D-   00                  BRK
  ⋮
0318-   00                  BRK   ABCDEFG
HIJKLMNOPQRSTUVWXYZ
 *
```

b) *300LL 〈CR〉

```
0300-   A9   C1          LDA #$C1  ⎫
  ⋮                                ⎬ 共40行
                                   ⎭
 *
```

c) *0300LLL 〈CR〉

```
0300-   A9   C1          LDA #$C1  ⎫
  ⋮                                ⎬ 共60行
                                   ⎭
 *
```

二、小汇编程序命令

因为小汇编程序（MINI-ASSEMBLER）是放在整数（INT）BASIC 文件中的，所以启动时需用INT命令，调出整数 BASIC，然后即可使用小汇编

注意　整数BASIC提示符是“>”

　　　浮点BASIC提示符是“]”

　　　监控提示符是“*”

　　　小汇编提示符是“!”

(1) 插入带INTBASIC文件的磁盘后，用以下操作即可进入小汇编：

] INT 〈CR〉　　　　此时APPLE Ⅱ用提示符>作为回答

>　　　　　　　　　　表示已进入INTBASIC控制之下

> CALL－151 〈CR〉 此时APPLE Ⅱ用提示符*作为回答

*　　　　　　　　　　表示已进入监控程序控制之下

* F666G 〈CR〉　　　此时APPLE Ⅱ用提示符！作为回答

！　　　　　　　　　　表示已进入小汇编控制之下

(2) 在规定内存地址送入符号指令

例如：在300单元送入符号指令CLD，则操作如下：

！ 300：CLD 〈CR〉　　APPLE Ⅱ汇编后将符号指令及机器码显示如下：

0300－　D8　CLD

！

又如：继续在301单元送入符号指令LDY #$FF，则操作如下：

！ 301：LDY #$FF 〈CR〉　　APPLE Ⅱ汇编后，将符号指令及机器码显示如下：

0301－　A0　FF　　　　LDY　#$FF

！

由于刚才已使用过300单元，现在是接着使用301单元，所以也可不打入301地址，而打入一个空格

！ ␣ LDY #$FF 〈CR〉

0301－　A0　FF　　　　LDY　#$FF

！

注意：①小汇编中不可使用符号地址

②操作数或操作数地址只能使用十六进制数，而且十六进制表示符号$可以省略不写

按以上操作用小汇编命令送入以下源程序时(程序起始地址是300)

LDA　#$C1

```
JSR    $FDED
CLC
ADC    #$01
CMP    #$DB
BNE    $0302
RTS
```

小汇编就把每条符号指令及其对应的机器码连同指令所在地址逐条显示出来。

注意：条件转移指令中的地址码部分，在小汇编中要用转移目标处的绝对地址不用符号地址也不用转移步长。

(3) 从规定的起始地址列出源程序及对应的机器码

例如要列出刚才打入的那段程序的符号指令及机器代码，则操作如下：

!$0300L <CR>

执行此命令的结果就会从指定的首地址0300开始显示满屏幕的符号指令及对应机器码(20行)。

注意：此处地址前的$不可以省略。

(4) 将一程序送入规定的存贮区域后，运行此程序。

例如对以上已送入300单元为首地址的这段程序，要使它运行，则操作如下：

!$0300G<CR>

注意：此处地址前的$不可以省略。

程序执行完毕后需要时可返回监控程序，用监控命令来检查程序运行结果，如检查某些内存单元内容和各寄存器内容是否正确等。

(5) 检查内存单元内容

例如要检查300单元的内容，操作如下：

! $0300<CR>

注意：此处地址前的$不可省略。

(6) 修改内存单元内容

例如要修改从300单元开始的连续5个单元内容，使皆为00，操作如下：

! $0300:00 00 00 00 00<CR>

注意：此处地址前的$不可省略。

最后说明一下BASIC、监控、及小汇编如何互相转入：

(1) 由BASIC转入监控

] CALL-151 <CR>　APPLE Ⅱ应以“*”作为响应

*

或：

> CALL-151 <CR>

*

(2) 由监控进入小汇编

* F666G <CR>　APPLE应以“!”响应

(3) 由小汇编返回监控

! $FF69G <CR>　APPLE应以“*”作为响应

(4) 由监控返回BASIC

*3D0G <CR>　APPLE应以“]”或“>”作为响应

(5) 由小汇编返回BASIC

! $03D0G <CR>　APPLE应以“>”作为响应

! INT <CR>　APPLE应以“>”作为响应

! FP <CR> 回到浮点BASIC，APPLE应以“]”响应

无论是在监控或小汇编，按RESET键都可回到BASIC。对没有使用DOS3.3 操作系统的机器，只能通过按RESET 键才会由监控返回BASIC。

三、编辑—汇编程序命令

用汇编语言编写好的源程序要送到计算机中执行，必须首先用机内的编辑程序（EDITOR）将源程序用ASCII码送入计算机，这个过程称为编辑，再由机内的汇编程序（ASSEMBLER）将已送入计算机的源程序翻译成目标程序将目标文件存入软盘，这个过程称为汇编，然后计算机才能运行该程序。在 APPLE Ⅱ 中编辑程序和汇编程序合并为一个文件称为 EDASM，下面介绍的就是EDASM各种命令的使用。

1．启动EDASM有两种方法，它们是：

（1）如果系统已经引导了DOS3·3，则应将DOS3·3软盘取出，换以TOOL KIT软盘插入驱动器，然后打入命令

] BRUN EDASM.OBJ ‹CR›

即可从磁盘调入EDASM的目标码文件，并开始执行这个程序，而使系统处于它的管理之下，屏上出现 EDASM 程序提示符:及游标:。

（2）如果系统电源接通以后没有在软盘驱动器内插入 DOS 3·3盘片，而是插入了TCOL KIT盘片。在这种情况下应打入命令

] RUN EDASM ‹CR›

这是从软盘上把EDASM的 BASIC 文件引入内存，然后开始运行它。屏上将出现：

APPLE Ⅱ EDITOR—ASSEMBLER

CURRENT ASSEMBLER ID STAMP IS:

P9—NOV—79 #×××××

字样，并有闪动游标，此时按一下 RETURN 键，待屏上出现EDASM提示符:和游标:时，即可使用。

2．如何使用编辑程序来编辑源程序。

（1）A命令(ADD)

使用此命令向APPLE Ⅱ 送入一个用汇编语言写的源程序，此

处以输入一个练习程序为例：

```
:A<CR>
1␣LDA #$AC; LOAD $AC INTO ACCUM.
2␣STA $06  ; AND INTO LOCATION $06
3␣BRK      ; RETURN TO MONITOR
4CTRL—D (或CTRL—Q)<CR>   则由A命令返回编辑
                         提示符
:
```

此处注意①每行的行号是 APPLE Ⅱ 自动显示的。②在各行写入的语句，若该行语句有标号则直接跟在行号后面打入语句，若该行语句无标号则应在行号后面打一个空格，然后打入该语句，这一个空格就留出了八个标号字符的位置。

操作举例如下

```
:A<CR>
1␣   ORG   $0300
2␣   CLD
3␣   CLC
4␣   LDA   $0350
5␣   ADC   $0351
6␣   STA   $0352
7␣HERE JMP HERE
8  CTRL-D<CR>
:
```

此处第 7 行有标号，所以不要打空格而应紧跟在行号 7 后面打入该语句。

(2) L命令(LIST)

使用此命令显示已送入APPLE的文本，操作如下：

:L　行号　<CR>

例如

:L　1-5<CR>　显示文本的1—5行

:L　<CR>　显示文本的所有行

(3) P命令(PRINT)

此命令用法同于LIST，但不显示行号。

(4) C命令(CHANGE)

此命令用来修改已送入APPLE机文本中的某行，操作如下：

:C行号·老字符串·新字符串<CR>

或者

:C行号$老字符串$新字符串<CR>

此处的·或$是分界符，不可省略。

然后APPLE Ⅱ 询问

ALL OR SOME? (A/S)?

因为一行中的老字符串可能不止一个，若要把该行中所有老字符串都修改为新字符串则回答A，若只把该行中的部分老字符串修改为新字符串则回答S。在回答S之后，对于要修改的老字符串就再按一次S键，对于要保留的老字符串则按ESC键。

操作举例如下：

我们用上面介绍的练习程序为例，欲把第1行中的数AC改成00，可以如下操作：事先应把该程序用A命令送入APPLE，然后

:C　1AC00<CR>

ALL OR SOME?(A/S)?

若按A键回答，APPLE修改完毕的第1行变为：

1 LDA　#$00　; LOAD　$00 INTO ACCUM.

若按S键回答后，再按一次S键和一次ESC键，则该行中的两个AC只有第一个被修改成00，第二个AC保持不变，即修改后的第1行变为：

1 LDA #$00　; LOAD　$AC　INTO ACCUM.

(5) D命令(DELETE)

此命令用来删除已用A命令送入 APPLE Ⅱ 的文本中的某行(或某连续段)。操作如下：

:D　行号 <CR>

例如欲把上面所用练习程序中的第一行删除，则应打入如下命令：

:D1<CR>

然后再用L命令检查，可以看到第一行已被删除。若是想删除连续的一段，例如把文本中原来的第2行到第3行都删除，则应接着刚才的删除之后再打入如下命令：

:D1—2 <CR>

再用L命令检查，可以看到文本中原来的第2行、第3行也被删除。

(6) I命令(INSERT)

此命令用来对已用A命令送入 APPLE Ⅱ 的文本在规定的行号处插入一行语句。操作如下：

:I行号<CR>

例如若想在前面使用的练习程序开始处插入一句：

ORG　$0300

则应按下面步骤操作：

:I 1<CR>

1␣ORG　$0300<CR>

2 CTRL-D<CR>　　　退出I操作

:

然后可以用L命令来检查一下插入结果。

(7) R命令(REPLACE)

此命令用来对已用A命令送入APPLE Ⅱ的文本中的某行(或某连续段)进行重写，例如对已插入ORG　$0300后的练习程序进

行以下操作：

:R3<CR>

3␣ADC　$09<CR>

4CTRL-D<CR>　　　　退出R操作

:

然后用L命令检查重写结果，就可以看到已把STA　$06改写成了ADC　$09

(8) F命令(FIND)

此命令用来在已用A命令送入APPLE Ⅱ的文本中找出某行或某连续段，或是找出某指定字符串所在行。

操作如下：

:F1<CR>

或

:F1—2<CR>

或

:F　B<CR>

注意：用F来寻找指定字符串所在行时，指定字符串的前后应加上分界符$或·。

(9) E命令(EDIT)

此命令用来对已用A命令送入APPLE Ⅱ的文本的某一行作局部编辑，在E命令控制下，又有以下几种操作命令表（表3—6），介绍如下：

这些操作命令的使用步骤如下：

:E 行号 <CR>

然后再具体对该行使用表3—6中所列各命令。

注意：若要由EDASM返回BASIC时，打入END命令即可。

3. 如何使用汇编程序来对源程序进行汇编

上面介绍的只是编辑程序的使用，对于一个已编辑好的源程

表3-6　　E 命 令 功 能 表

字　符	功　能
⟶	游标向右移动一个位置
⟵	游标向左移动一个位置
RETURN	接受对该行的编辑命令并退出E命令回到EDASM
CTRL-D	在游标指示的位置上删除字符
CTRL-I	在游标指示的位置上插入字符
CTRL-T	把游标指示位置以后的字符全删去
CTRL-R	按游标指示的位置进行改写，不需改写的字符用⟶键跳过去
CTRL-F	在此命令后跟着打入要在该行中寻找的字符，则游标就会自动移到所寻字符位置处
CTRL-X	退出E命令，并使正在进行的操作无效，回到EDASM

序如何使用汇编程序翻译成目标程序再使之运行，现在把步骤介绍如下：(我们此处介绍的步骤是针对使用一台软盘驱动器，而且TOOL KIT文件和用户程序都使用一张盘片的情况)

(1) 首先把已编辑好的用户源程序文件存入盘片，存盘时要给源程序起个名字。例如要把上面的练习程序进行汇编，首先给它起个名字EXE1，用此名字存盘

:SAVE EXE1 <CR>

存盘之后可用CATALOG命令检查是否已存进盘中。

:CATALOG <CR>

可以看到盘中已有一个命为EXE1的文件了。

(2) 用以下操作对已存在盘上的EXE1文件进行汇编

:ASM EXE1<CR>

此命令执行时，系统首先从EDASM文件中调出汇编程序，并在屏上显示出如下字句：

PRESS ANY KEY TO CONTINUE

这时应压任何一个键作为回答，APPLE Ⅱ就开始对盘上的EXE1文件进行汇编，等到汇编完毕时屏上又再次出现如下字句：

PRESS ANY KEY TO CONTINUE

这时再压一次任一个键作为回答，则EDASM文件中的编辑程序又被调出，屏上回到EDASM提示符

:

至此汇编完毕，然后可用CATALOG命令，看到盘上已有一个名为EXE1.OBJ0的目标码文件。

附带说明一下，在EDASM控制下CATALOG，SAVE及LOAD命令都可使用。前两者刚才已经使用到，而LOAD命令是在要调出已存盘的源程序文件时用。例如用命令

:LOAD EXE1〈CR〉

就可把已存在盘上的EXE1文件调进存贮器中。

若想在汇编过程中用打印机印出清单文件时，应先打开打印机电源并应在ASM EXE1命令前加一句

PR#1, L××P××CTRL－I80N 〈CR〉 的命令。

此处的L××应为1页中要打印的行数，而P××应为一页中的总行数。例如我们在L之后填15，P之后填20，就表示这一页共20行，打印15行后就空5行。

:PR#1, L15P20CTRL－I80N 〈CR〉

:ASM EXE1 〈CR〉

第一句中的P××中填的数若小于L××中的数，则打印时就不分页了，而是连续打印下去。

(3) 如何运行已经汇编好的目标码程序

仍以EXE1为例，现在是要运行它的目标码文件 EXE1.OBJ0

此时先使用 END 命令由EDASM返回 BASIC，当屏上出现BASIC提示符时，再用BASIC命令运行此程序。

]BRUN EXE1.OBJ0 <CR>

程序就运行了。运行完毕回到监控，当由监控再次运行此程序时，就可用以下命令：

*0300G <CR>

这就不用再调盘，而直接运行已送入存贮器中的目标程序。

当遇到源程序汇编后产生的目标文件不止一个时，例如第二章例5三个数据块传送的程序中，由于程序由三个ORG伪指令语句分成了三段，因此将这个源程序送入APPLE Ⅱ 用名字DBLOCK汇编之后就产生了三个目标文件，它们是DBLOCK.OBJ0，DBLOCK.OBJ1及DBLOCK.OBJ2。那么在运行此程序时，应首先在BASIC提示符]下用BASIC命令

]BLOAD DBLOCK.OBJ0 <CR>

]BLOAD DBLOCK.OBJ1<CR>

]BLOAD DBLOCK.OBJ2<CR>

把这三个目标文件都调入内存，再转入监控程序，先用监控命令确定6000～6063，6100～6103，6200～6263三段存贮区域的内容，然后再用监控命令

*0300G <CR>

就可使程序得到正确运行。这可用检查7000～7063，7100～7163，7200～7263三段存贮区域的内容和6000～6063，6100～6163，6200～6263三段存贮区域内容是否相符来判断程序执行是否正确。

§3—5 AIM-65监控程序，文本编辑程序及汇编程序的使用方法

一、监控程序各种操作命令

首先说明一下，RESET命令和ESC命令的操作内容。

RESET 命令 ——进入监控程序并完成监控程序所需要的初始化工作。

ESC命令——重入监控程序。

在使用监控程序时要注意，它的提示符是<，按RESET键或ESC键都可使机器处于监控程序控制之下，在显示器上显示出监控程序提示符<。在以下所述各种命令的操作失败时，可使用RESET或ESC键使回到监控程序控制之下，然后再重新开始操作。

1．存贮器测试命令

(EXAMINING MEMORY)——M

在按了M键之后，再用键送入程序员所需要测试的内存单元地址（一般为四位十六进制码），最后按 RETURN 键（或空棒SPACE），显示器上立即显示出被选中的内存单元内容，而且显示的并不只是这一个单元的内容，而是从这个单元开始的连续四个单元内容。

如果需要连续测试后继单元的内容，则只要连续按SPACE 空棒即可。

2．存贮器修改命令(CHANGING MEMORY)——/

程序员若需要修改某一内存单元内容时，应首先用如上所述的M命令测试该单元内容，然后用/命令进行修改。即先按/键，随后依着被选中的四个地址单元顺序用键送入需改动的内容。若是对其中某个单元内容不需改动，那么只要按下SPACE 棒就可以保持其原有内容不变。如果需要继续修改紧跟着的后续四个单元内容，只要再按一下/键就可进行下四个单元的内容修改，而按下RETURN键就可终止修改内存的操作。

3．寄存器测试命令(EXAMINING REGISTERS)——R

这个命令可以用来测试当前状态下程序计数器PC，状态寄存器P，累加寄存器A，变址寄存器X，变址寄存器Y及堆栈指针S寄存器的内容。

只要用键打入R命令后，打字机立即打出一排字符，对应这六个寄存器的名字。即：

* * * *　　PS　　AA　　XX　　YY　　SS

紧接着第二排就打印出并同时在显示器上显示出这个寄存器对应的内容。注意上面的* * * *代表的是程序计数器PC，它的内容应当是四位十六进制码，而状态寄存器PS，累加器AA，变址寄存器XX和YY的内容则都应当是两位十六进制码，堆栈指针SS因规定是在1页，所以只显示低8位，因此也是两位十六进制码。

4. 寄存器修改命令(CHANGING REGISTERS)

需要修改某个寄存器内容时，可以首先按上述方法用R命令测试寄存器，然后再按下所需要修改内容的寄存器名字，最后再送入修改内容(四位或两位十六进制码)，至此修改寄存器的操作即告完成。修改寄存器的操作亦可单独进行而不必跟在测试寄存器的操作之后。

注意：在按下需修改其内容的寄存器名字时，代表符号（即按哪个键）和实际的寄存器之间对应关系为：

程序计数器PC —— *
累加寄存器A —— A
变址寄存器X ——X
变址寄存器Y ——Y
堆栈指针S ——S
状态寄存器P ——P

其中修改PC时需按RETURN键来结束修改操作，其它寄存器修改不需按RETURN键。

在修改寄存器操作完成之后，可再使用测试寄存器命令 R，显示器上便可显示出修改后的新内容。

5. 使用打印机命令(USING THE PRINTER)

—CTRL PRINT

同时按下CTRL和PRINT两个键，可使打字机处于开或关状态，如果打字机原状态是“关”，则同时按这两个键就使打字机改变为“开”状态。如果打字机原状态是“开”，则同时按这两个键就使打字机改变为“关”状态。

按这两个键时，打字机的状态在显示器上用OFF或ON显示出来。

注意：AIM-65在开启电源时，打字机一定是自动处于“开”状态的。

使打字机空走一行的操作是按下LF键。

6．初等汇编命令或称符号指令输入命令

(INSTRUCTION MNEMONIC ENTRY)——I

之所以把这种操作命令称为初等汇编或称为符号指令输入命令，而不把它称为汇编命令，这是因为这种命令虽可把送入的符号指令逐条翻译成机器指令并送入相应的地址单元中，但在符号指令中不允许使用符号地址，也不允许使用伪指令。它的翻译过程没有两遍扫描的过程，也产生不了符号表和清单，这都是和一般的汇编程序操作命令不同的，也就是说这种操作不是在汇编程序支配之下进行的，而是在监控程序支配之下的一种对符号指令逐条翻译成机器指令的操作。这种功能对于调试简单的程序是极为有用的。下面我们来具体说明I命令的操作步骤：

首先按下I键，接着按＊键，并把待输入的符号指令程序起始地址用四位十六进制码输入（我们已知道这是修改程序计数器PC的操作），最后按下RETURN键，下面就可以在显示器所指示的地址单元上跟着游标∧位置逐条输入符号指令了。程序输入完毕时可用ESC键终止操作。这里要注意，当输入的符号指令有错误时，（即不符合6502对符号指令的规定时）显示器上出现的仍是本条指令所在的地址单元码，地址并不下滑，这是机器要求重新送入正确的指令。如果在送入符号指令的过程中发现有错时，可用

DEL键进行删除和重新送入正确的符号。

以上操作如把按I 键和送入程序起始地址两个步骤对调一下也是可以的。

我们可以细致一点说明I命令操作过程中要注意的几个问题：

(1) 程序计数器必须置于待输入程序的起始内存单元。

(2) 所有单字节的指令码在第三个字符输入后立即被“汇编”。

(3) 包含两个或三个字节的指令码要由 RETURN 键来结束输入。

(4) 输入过程中发生错误时，用下面办法之一纠正错误：

1) 用DEL (删除) 键进行删除 (CTRL-H键也具有同样删除功能)

2) 用修改程序计数器内容的办法，重置程序计数器于输入出错的那条指令所在内存地址上，再用I命令重新开始。

3) 用ESC键回到监控程序提示符，重新开始。

(5) ESC 键可以用来退出这种方式，即终止符号指令输入操作。

(6) 指令所在地址码和操作数所在地址码一律被看作是十六进制而不必加十六进制表示符号 $。

如果待输入的程序是由机器指令构成，则用前面说过的修改内存命令/来进行是很简单的，但是对于调试和执行程序来讲，用符号指令输入命令 I 来输入由符号指令构成的程序则是更为方便的。对于比较长的程序，还需要用汇编程序操作命令来对源程序进行汇编，这点我们在下面将加以说明。

7. 反汇编命令 (DISASSEMBLE MEMORY)——K

反汇编又可称为逆汇编，它的操作过程是把内存中的十六进制目标码转换为符号指令码，操作如下：

首先按下K键，并在由于按下K键显示器上出现的“* =”之

后送入需要反汇编的内存首地址，随着按下RETURN键，AIM-65收到这个首地址后立即响应，在显示器上出现/符号，此时程序员应送入需反汇编的指令条数（十进制数）。例如，本次需反汇编15条指令，那么就在/符号之后用键送入15两个十进制码，AIM-65就立即开始反汇编过程。

对于反汇编操作需要注意的是：

（1）在/符号后可送入的指令条数应在十进制数01～99之间，而数00则表示100条指令之意（必须送入两位十进制数，当指令条数<10时，十位数字上送入0）。若是不送具体数而按下RETURN键，则只反汇编一条指令。若是按下·键或SPACE键，则反汇编过程将连续进行下去，直到按ESC键时才告终止。

（2）对于内存单元中不符合机器指令规定的那些十六进制码，反汇编的结果找不到对应的符号指令则用问号？来表示。将反汇编指令K和符号指令输入命令I结合起来使用将带来许多方便。例如，先用I命令送入了一段用符号指令构成的程序，然后想要再把这段程序仍用符号指令打印出来的话，就可使用K命令来完成。当然要注意两者的内存地址必须相符合。

8. 执行程序的命令（G COMMAND）——G

当利用上述已讲到的命令把一段编好的程序送入内存后，就可使用G命令使这段程序运行。首先要利用修改程序计数器PC的命令＊来把PC置于这段程序的启动地址，然后按下G键并随之按RETURN键就可使程序运行。当程序运行完毕时，显示器显示下一条指令的地址及内容。

至此我们所了解的这些操作命令足可以用来进行简单的程序调试了。但是在这些程序中不能使用标号地址和伪指令，所以内存单元的分配工作还要程序员自己来进行。因此对于复杂一些的程序进行调试，还需使用文本编辑及汇编等操作命令。

9. 如何使用盒式录音机转贮目标程序和数据

AIM-65允许配有两台普通的盒式录音机做为磁带机，以用来记录和读出目标码或源码，此处我们只说明如何对目标码进行记录和读出。关于源码即ASC II字符的记录和读出，将在下面再加以说明。

录音机和AIM-65之间的连线方法是把AIM-65机器后方的插头上标M字样的腿接到录音机MIC处，就可以实现转贮操作，把AIM-65后方插头上标L字样的腿接到录音机EAR处就可以实现读带操作。

(1) 如何将内存中的目标码记录到磁带上去，转贮操作命令(DUMP COMMAND)——D

首先按下D键，AIM-65响应后就立即询问需要存到带上去的目标码是在内存地址的什么区域内，显示器上显示“FROM =”此时应打入内存起始地址并按下RETURN键，则又跟着显示“TO =”应再打入内存终了地址和按RETURN键，显示器跟着问询“OUT =”此时应按T键做为回答，表示这段目标码是要送到带上去。显示器又接着问“F =”这是要求程序员回答录到带上去的文件名字是什么，可以用少于或等于五个字符作为文件名送入，AIM-65得到文件名后还要提出最后一个问题，显示器上出现“T =”这是问用哪一台录音机，1号还是2号，一般只接一台，可以回答1号，即用键按下1即可。注意这时不要按RETURN键，而要使录音机工作在录音状态，当录音机转速稳定后(一般走十个字即可使转速达到稳定)再按下RETURN键，录制工作就立即开始了，此段目标码录制接近结束时显示器又问MORE？一般情况可回答N，那么录制操作就告结束，AIM-65又回到监控程序提示符<。

现在我们顺着操作步骤再叙述一遍，用方框标明的地方是要求程序员用键送进去的，而未加标注方框的则是AIM-65显示器上显示的字样。

D

FROM = ×××× ‹CR›　　TO = ×××× ‹CR›

OUT = T　　F = ×××××　　T = 1 ‹CR›　　××

MORE?　　N

下面我们再交待一下注意事项：

1) 在F=后面送进的文件名，如果是五个字符则可不按‹CR›（RETURN 键之意）如果少于五个字符则要按‹CR›。

2) 在录制过程中××处显示的是不断更换的00，01，02，——这是对被录的数据块计数，80个字节为一个数据块，录制的时候是以数据块为单位录到带上去的。

3) 录制过程的结束必须是在MORE?后回答N来结束，如果用ESC键结束则最后一个不满80个字节的数据块就不能录到带上去。

4) 在MORE?后若是回答Y，这里表示本文件的内容除了已送进AIM-65的FROM=××××TO=××××内存区域的目标码外，尚有其它内存区域的内容需要录制，因此在用Y做了回答后，AIM-65又询问FROM= ×××× ‹CR›

TO = ×××× ‹CR›，这次之后如再无内容需要录制了，那么在又问MORE?时应回答N做为结束。

5) 应该很好地注意使录音机的操作和AIM-65的操作互相配合好。

例如：在内存上的信息记录到带上之前，应使录音机在录音状态已工作十个字以上（指录音机上的计数器指示值）以使转速达到稳定。

还要注意，每两个文件在带上的距离要相距十个字以上。此外为了在磁带上迅速正确找到被记录的文件所在位置，应正确使用录音机上的计数器装置，最好做一张表格，记录下文件名，起始内存地址，录音机上计数器的数字，以备查询方便。

6) 如果在转贮命令D的操作步骤中，在OUT＝之后回答的不是T，而是P或‹CR› (RETURN)，这表示内存某段区域中的目标码是要被送到打字机或显示器而不是送到带上，那么AIM-65就立即执行命令而不再问询文件名和录音机台号，其它操作步骤相同。

最后我们还要交待一下，本份操作说明中所提到的磁带转储及读出命令的操作都是按照AIM-65记录格式进行的，至于采用KIM－1 磁带记录格式的情况如何操作，我们此处就不作介绍了。

(2) 如何读出磁带中记录的目标码，(LOAD MEMORY)——L 首先按下L键，AIM-65响应后立即问询“IN＝”意即问询目标码从什么设备进入AIM-65内存，此处应回答T，接着又问文件名“F＝”应回答欲从带上读进内存的文件名，最后问录音机台号“T＝”按下1键作为回答，但不要按‹CR›等录音机做好准备，把带置于被查找文件前5～10个字并使录音机置于放音状态，然后再按下‹CR›，这时就开始读带过程了。当文件找到后，AIM-65就开始读文件的过程，这时显示器上显示的字样是LOAD F＝××××× BLK＝×× BLK后×× 显示的是不断更换的00，01，02，… 表示正在读出的数据块号。读毕，AIM-65回到监控程序控制下，显示器回到监控程序提示符＜。

现在我们顺着操作步骤再叙述一遍，用方框标注的地方是要求程序员用键送进去的，而未加标注方框的则是AIM-65显示器上显示的字样。

L

IN = | T |　　F = | ×××××× |　　T = | 1 〈CR〉 |

LOAD　　F = ××××××　　BLK = ××

下面我们再交待一下注意事项：

1) 上面第二行F＝后面送进去的文件名如为五个字符则不需按〈CR〉，如少于五个字符则需要按〈CR〉。
2) 在T＝之后按下1键但不要马上按〈CR〉，等录音机已用计数器找到带上存该文件的相应位置使录音机处于放音状态时才能按下〈CR〉。
3) 操作过程中注意录音机的操作和AIM-65的操作配合好。
4) 文件读进内存哪一段区域是和文件录制过程时规定的内存区域相一致的。
5) 读文件过程开始之后，如果显示器上显示的不是上面第三行显示的LOAD F＝×××××× BLK＝××而是SRCH F＝×××××× BLK＝××这是说明正在读的文件不是程序员要找的文件，而是另一个文件，此处跟在SRCH后的文件名不是程序员要找的文件名，而是正在读的文件名。

10.执行/跟踪命令（EXECUTION/TRACE COMMAND)

以上在第8个问题中曾说明了G命令如何使一个已送入内存中的一段程序运行，但在那里我们少交待了一句话，即那里的G命令执行是在AIM-65处于RUN状态下进行的，AIM-65显示器的左下方两个小圆盖下面有两个开关，左边那个开关向下打是使AIM-65处于RUN状态(意即连续运行程序)，开关向上打是使AIM-65处于STEP状态（意即步进运行程序)。

当AIM-65处于STEP状态时，有以下几种执行/跟踪命令，使

程序运行可以步进式进行，这对程序调试带来许多方便。

执行/跟踪控制命令如下：

Z命令——触发指令跟踪方式打开或关闭
(TOGGLE INSTRUCTION TRACE MODE ON/OFF)

V 命令——触发寄存器跟踪方式打开或关闭
(TOGGLE REGISTER TRACE MODE ON/OFF)

H 命令——跟踪程序计数过程（TRACE PROGRAM COUNTER HISTORY)

? 命令——显示断点 (DISPLAY BREAKPOINTS)

#命令——清除断点 (CLEAR BREAKPOINTS)

B命令——设置断点 (SET/CLEAR BREAKPOINTS)

4命令——触发断点方式打开或关闭 （TOGGLE BREAKPOINT ENABLE ON/OFF)

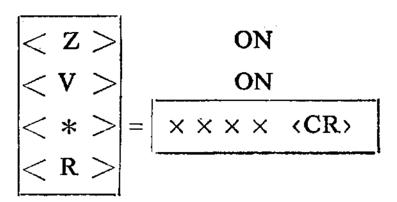
G命令——从程序计数器 PC 的数值开始执行程序（START EXECUTION AT PROGRAM COUNTER ADDRESS)

以上命令中的 Z，V 和 4 命令含义说明中的所谓“触发”是指按下该键时若显示 OFF 那么再按一下此键就会显示 ON，表示对应的方式已经打开，若需关闭该方式时只要再按一下此键就会显示 OFF。

以上这些命令只在AIM-65处于 STEP 状态时才有效，并且注意在使用这些命令运行程序之前，就应把程序送入内存。对于如何使用这些命令我们作以下说明：

(1) 同时采用指令跟踪及寄存器跟踪。

步骤如下：



* * * *　PS　AA　XX　YY　SS

××××　××　××　××　××　××

<G>/ •

注意标注方框处是要求程序员打进去的命令及地址，在G命令之后应随之再按一下 • 键或SPACE键而不要按‹CR›（若按‹CR›则只执行一条指令）。在G命令之后亦可不按·或SPACE而送进两位十进制数以表示要运行的指令条数，这样操作之后程序运行过程中就会自动显示并打印出寄存器跟踪和指令跟踪的情况，便于程序员查看程序运行所经过的途径及每条指令执行后的各寄存器状况。

请注意在以STEP方式运行程序时，BRK指令是不能停止程序运行的，在G命令后送·或SPACE时需用ESCE停止程序运行。

（2）仅采用指令跟踪

步骤如下：

< Z >　ON

< V >　OFF

< * > = ×××× ‹CR›

< R >

* * * *　PS　AA　XX　YY　SS

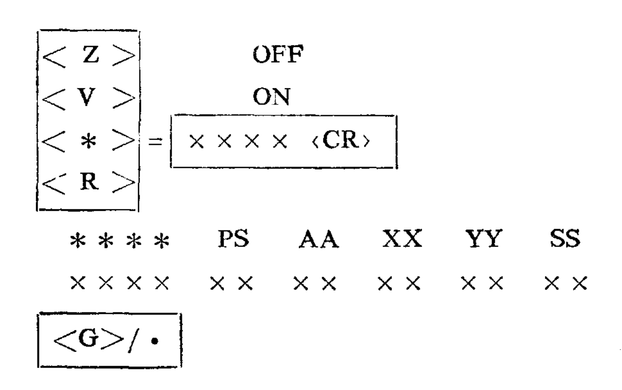
××××　××　××　××　××　××

< G >/ •

如此操作后可以发现打印机记录下来的只有指令跟踪的过程而无寄存器跟踪的过程。

（3）仅采用寄存器跟踪

步骤如下：



如此操作以后可以发现打印机记录下来的仅有寄存器跟踪过程而无指令跟踪过程。

(4) H 命令

H命令可以分别和以上三种方式连用，即在程序运行结束后，按H键 AIM-65 便会自动打印出刚刚执行过的四条指令地址及下条指令地址，便于程序员查看。

(5) 断点跟踪命令

？、#、B、4 这四个命令都是有关断点设置的命令。

1) B 命令：它是用来设置断点的，AIM-65 允许在一个程序中最多设置四个断点，操作如下：

在程序已被送入内存之后，按下B键，则显示出BRK/字样，此时程序员应按照选好的断点地址顺序，分别在/下打入 0，1，2，3的数字，然后再分别送入这四个断点的地址。例如：

<B> BRK/0 = 0310‹CR›

<B> BRK/1 = 031B‹CR›

<B> BRK/2 = 0328‹CR›

<B> BRK/3 = 0‹CR›

这就在程序中设置了第 0 号断点地址为0310，第 1 号断点地址为 031B，第 2 号断点地址为 0328，第 3 号断点地址为 0000

(即只设置了前三个断点，第四个断点未设置）设置好断点地址之后，再按前面所述方法选择好适当的指令跟踪和寄存器跟踪方式使这两个跟踪开关或者都打开，或者打开一个，或者都关上，然后使程序运行，那么程序运行过程中遇到断点地址时就会自动停止运行，此时程序员可以运用R命令，M命令来检查有关寄存器及内存单元内容，以观察程序执行情况，然后可以用G/·命令使程序继续下滑，等遇到断点时将再次停机。

2）#命令：它是用来同时清除四个断点的。只要按下#键，显示器上便会出现<#>OFF字样，此时四个断点全被清除，即四个断点地址全被变成0000。

3）?命令：它是用来显示四个断点地址的，只要按下?键，显示器上便会同时出现四个断点的地址。

4）4命令：它是用来控制使断点生效的开关，当4键按下出现< 4 >ON字样时，所设的断点才能在程序运行过程中生效，若是处于< 4 >OFF状态，则所设断点在程序运行过程中无效，只要反复按4键就会使ON和OFF状态交替反复变化。

那么上面在讲B命令时，已经说了设置断点后可选取适当的跟踪方式运行程序，在程序运行过程中遇到断点就会自动停止运行，现在对这段话我们还要再补充一句，就是在设置好断点地址后还必须使4开关处于ON状态才行。

二、文本编辑程序各种操作命令

我们知道用汇编语言所写的程序要能够在计算机中执行，必须首先输入用ASCII字符书写的源程序，再由机中的汇编程序把源程序汇编成目标程序并送入相应内存单元之中，然后计算机才能运行这一程序。文本编辑程序就是用来供程序员用键输入ASCII字符书写的源程序的，顺便说一句，文本编辑程序不仅可以用来输入和编辑各种语言的源程序用，而且可以输入和编辑文

章，因为源程序也好，文章也好，它们都是由ASCII字符构成的，我们可以统称为文本。

1．AIM-65文本编辑程序操作命令可以分成四类

(1) 编辑程序进入和退出命令

E命令——进入编辑程序并完成编辑程序所需的初始化工作

T命令——重入编辑程序并显示头行，如果已在编辑程序中，则T命令可显示头行。

Q命令——退出编辑程序并重入监控程序

ESC命令——重入监控程序

(2) 文件输入/输出和修改命令

R命令——从输入设备读若干行内容进文本缓存区

I命令——插入一行

K命令——删去一行

L命令——从已放好一个文本的文本缓存区中把指定行数的内容送到某输出设备

(3) 行指针位置调整及显示

T命令——把行指针移到文本顶部并显示第一行内容

B命令——把行指针移到文本底部并显示最末行内容

U命令——行指针上移一行并显示该行内容

D命令——行指针下移一行并显示该行内容

(4) 字符串查找和修改

F命令——查找字符串，从行指针指向的当前行开始向下搜索程序员所指定查找的字符串，找到后并加以显示。

C命令——修改字符串，首先从行指针指向的当前行开始搜索程序员所指定的旧字符串，然后再用程序员指定的新字符串来更换掉旧字符串

现在我们说明一下，上面所提到的文本缓存区和行指针的含

义是什么。

文本缓存区是指由程序员在进入编辑程序时指定的一块内存区域，这块内存区域用来存放待编辑的文本。应当注意的是文本缓存区不允许使用0页和1页，因为0页已为系统软件占用，而1页为堆栈。

行指针的含义是什么呢？我们知道AIM-65文本编辑的各种命令都是和行有关的，所有的编辑操作都是从当前行开始，当前行就是用行指针来识别的，行指针总是指向当前行第一个字符之前的位置，编辑命令执行之后，行指针就会下移1行或是移到文本最末处，这要看执行的是什么命令而定。

行指针的位置可以由程序员移动，这由行指针调整命令（T，B，U，D命令）来完成。

2．操作举例

对于以上各种编辑命令究竟如何使用，我们还是用实例来说明为好：

下面是一段用户程序，这个程序是用来把8位二进制小数（即两位十六进制小数）转换成三位十进制小数（BCD码），原始二进制小数存放在BIN单之中，结果十进制小数存放在BCD，BCD+1，BCD+2，三个单元中，MU10是乘10子程序入口，关于这个程序的算法在第二章已述此处只是讨论如何用EDITOR把源程序正确无误的送入内存的文本缓存区中。

程序员编好的源程序如下：

```
; PRO1
BIN = $0300; PRIMARY
        ; DATA ADDRESS
BCD = $0301; RESULT
           ; ADDRESS
MU10 = $0240; *10 ROU-
```

我们给该程序起名为PRO1

（1）使用E命令把这个源程序用键送入内存文本缓存区中，使在文本缓存区中形成源程序文本，操作如下：

```
        ; TINE
* = $0200
LDY   #00
LDA   BIN
STA   $01
NEXT JSR MU10
STA   BCD, Y
LDA   $01
INY
CPY   #03
BNE   NEXT
BRK
* = $0240
MU10   LDX   # = 00
STX   $00
LDX   #09
CLD
LOOP1   CLC
ADC   $01
BCC   LOOP2
INC   $00
LOOP2   DEX
BNE   LOOP1
STA   $01
LDA   $00
```

```
<E>
EDITOR
FROM = 0400<CR>
TO = 0550<CR>
IN = <CR>
```

标注方框处是程序员需按键之处。

做完这些操作后就可跟随显示器上的游标∧把源程序的每个字符逐个用键输入。

注意每行字符输入完毕时需按‹CR›键。

上面操作中的FROM = ××××

TO = ××××就是程序员所指定的文本缓存区地址范围，我们已经说过，它不能设在0页和1页。

在IN = 后面按‹CR›键的意思是从键盘输入。

在用键盘输入源程序时请注意：

1) 在最后一行 RTS 送完后应按二次‹CR›表示结束，待显示器上出现END字样时再按Q键退出编辑程序。

RTS

2) 输入指令时，按指令格式该留空处应按SPACE 键以留空，如果没有留空将来在汇编处理过程中就将被当作出错处理。

3) 输入字符过程中按错了键时可用DEL 键进行删除。

我们应当注意到E 命令是为形成新文件用的，而以下所述各命令是用来处理已在文本缓存区中形成的文件。

(2) 使用T命令和L命令把已在文本缓存区中形成的源程序文本送到显示器和打印机，操作如下：

| <T > | 重入编辑程序并使行指针指向文本顶部。

; PRO1　AIM-65 显示出第一行内容。

| L | 文本输出命令。

/ | • | 输出多少行在／之后可跟具体数字亦可跟•字符。

OUT = | 〈CR〉 | 文本输出到显示器和打字机，如果只送到打字机此处就按P键。

这样操作之后，整个源程序的内容就会逐行送到打印机上去。

(3) 使用行指针位置调整命令来显示在文本缓存区中的源程序的各行内容。

操作如下：

按T键使显示器显示第一行内容　PRO1

按B键会使显示最后一行内容　RTS

再按U键就会显示上一行内容　LDA　$00

继续按U键将显示STA　$01

接着按D键则会显示下一行内容；LDA　$00

由此就可了解T，B，U，D四命令是如何调整行指针位置的。

(4) 如果在用E命令编辑源程序文本的过程中由于不小心按错了键使存入文本缓存区的源程序发生了错误成了如下所示的样子，那时对已形成的有错误的旧文件应如何进行修改？

```
PRO1
BIN = $0300 ; PRIMARY
            ; ADDRESS
BCD = $0301; RESULT
            ; ADDRESS
MU10 = $0240; *10 ROU—
              ; TINE
 * = $0300
LDY   #00
LDA   BIN
STA   $01
NEXT  JSR   MU10
STA   BCD, Y
LDA   $01
INY
CPY   #03
BNE   NEXT
BRK
 * = $0240
MU10  LDX   #00
STX   $00
LDX   #09
LOOP1  CLC
```

把这个有错的源程序和正确的源程序比较一下，我们可以发现在第一行、第三行、第八行处有错，还可以发现少了CLD和RTS两行，此外在倒数第四行处BNE LOOP 1 错成为BNE LOOP2一共六处错误。

下面我们把使用F、C、K、I命令来改正这个源程序错误的操作过程列出并加以说明。

1) 把第一行PRO1改为；PRO1是在使用重入编辑程序命令T之后，显示出第一行内容PRO1后用K（删除一行)和I（插入一行）命令完成的。注意K命令是删除当前行，删除完毕后显示下一行，而I命令是插在当前行的前一行，按下I键之后就应随着游标位置送入欲

```
ADC   $01
BCC   LOOP2
INC   $00
LOOP2  DEX
BNE   LOOP2
STA   $01
LDA   $00
```

插入的字符串；PRO1 用〈CR〉结束插入操作，插入完毕后显示下一行。

2）把第三行ADDRESS改为DATA ADDRESS 这是用D命令使行指针下移一行并显示该行内容；ADDRESS 然后用K命令删除该行，删除完毕显示下行内容 BCD = $0301；RESULT再使用 I 命令在这行之前插入；DATA ADDRESS〈CR〉插入完毕显示下行。

3）把第八行 * = $0300改为 * = $0200，这是用C（修改字符串）命令完成的，按下 C 键之后，随着游标位置送入待修改的字符串 * = $0300〈CR〉AIM-65 查找到这行后就显示该行内容 * = $0300此时应再按一次〈CR〉并跟在TO = 字样之后送入正确的字符串 * = $0200〈CR〉AIM-65完成修改操作后显示出修改后的字符串 * = $0200。



PRO1

= <K>

PRO1

BIN = $0300；PRIMARY

= <I>

；PRO1〈CR〉

BIN = $ 0300; PRIMARY

= <D>

; ADDRESS

= <K>

; ADDRESS

BCD = $ 0301; RESULT

= <I>

; DATA ADDRESS ‹CR›

BCD = $ 0301; RESULT

= <C>

* = $ 0300 ‹CR›

* = $ 0300 ‹CR›

TO = * = $ 0200 ‹CR›

* = $ 0200

= <F>

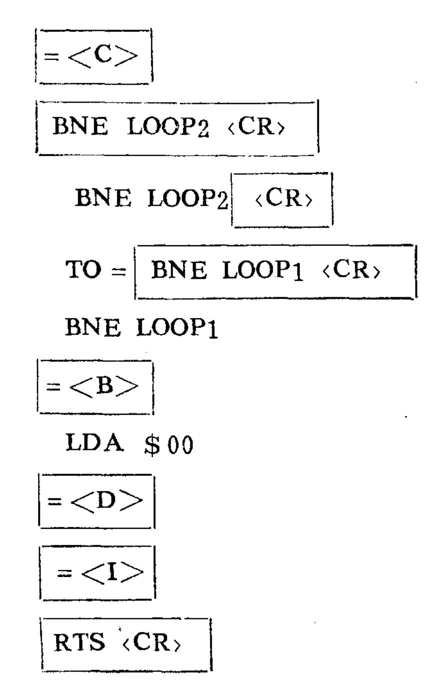
LOOP1 CLC ‹CR›

LOOP1 CLC

= <I>

CLD ‹CR›

LOOP1 CLC



4）把漏了的CLD这一行补上，这是用F(查找字符串)命令找到下一行字符串LOOP1 CLC,然后再用I命令把CLD插入完成的,按下F键后随着游标位置送入待查找的字符串LOOP1CLC ‹CR› AIM-65完成查找操作后,把找到的这一行LOOP1 CLC显示出来,此时再使用I命令在LOOP1 CLC这行前面插入一行CLD。

5）把BNE LOOP2改为 BNE LOOP1 方法同把＊= $0300改为＊= $0200。

6）把漏了的RTS这一行补上，此处是用B命令把行指针移到底部并显示最后一行LDA　$00然后用D命令使指针下移一行,再接着使用I命令插入RTS,但因此处是在文本底部操作,所以D及I命令之后并不显示下行内容(文本已无下行内容了)。

至此全部错误修改完毕，为了检查经过F、C、K、I等命令编辑修改之后的源程序是否已正确无误，我们可以再使用T、L命令用打印机打印出经过修改的整个源程序文本。

仔细检查一下就可以发现文本编辑程序的各种操作命令中尚有R命令如何使用还没有具体说明，这些我们留待下面再作补充。

3. 如果在E命令编辑一个源程序文本的过程中，由于程序员规定的文本缓存区范围小了，不够存放文本，AIM-65显示出END字样时应如何处理？如果采用使本次编辑操作作废的方法重新开始编辑，重新由FROM＝×××× TO＝×××× 规定新的文本缓存区范围，那么显然这对一个文本已接近编辑完毕的情况是不合适的。

为了采用有效的可以接着刚才的结束处继续向下编辑的办法，我们首先应当了解一下文本缓存区的上限、下限地址存放在哪些内存单元，我们可以联想到只要修改一下存放文本缓存区上限地址的内存单元内容，使上限扩大不就行了吗？

按AIM-65规定，这些内存单元是：

00E1：已使用的文本缓存区结束处地址低位

00E2：已使用的文本缓存区结束处地址高位

00E3：程序员规定的文本缓存区下限地址低位

00E4：程序员规定的文本缓存区下限地址高位

00E5：程序员规定的文本缓存区上限地址低位

00E6：程序员规定的文本缓存区上限地址高位

因此我们只要修改00E5和00 E6 两单元的内容以扩大文本缓存区上限地址就行了。

下面我们来举例说明：还是以小数二——十转换程序 PRO1为例，如果原来规定文本缓存区地址范围是0400～0500，在用 E命令输入源程序的过程中发现输入到CLD这一行处显示器上就出现END字样，表示不能再输入了。

这时可用Q或 ESC 命令退出编辑程序回到监控程序，并用M命令检查 00E1～00E6 这几个单元的内容如下：可以知道源程序编辑

<M>＝00E1 FA 04 00 04 到CLD这一行时，文本缓存区

< >00E5 00 05 FA 04 已用到04FA地址了，我们现在

<／>00E5 50 05 用／命令把存放文本缓存区上限地址的00E5，00E6两单元内容由0005修改成 5005 亦即把上限地址由0500修改为 0550 然后再用T命令重入编辑程序，用B命令找到文本底部再用 D命令使指针下移一行，接着就可以反复使用插入命令I把未输完的源程序全部输入完毕。最后我们再说明一下程序员应如何确定文本缓存区范围为多大，也就是说文本缓存区范围应如何计算，我们知道文本的编辑是对ASCII 字符进行的，每一个 ASCII 字符占一个字节，即一个内存地址单元，把源程序中所有的字符及每行回车符〈CR〉所占地的址加起来，还要再留出若干地址单元供修改错误及插入新行用，程序员根据这个原则就可给出文本缓存区的大致范围。

4. 如何把经过编辑的文本内容转贮到磁带，又如何由磁带读回到内存。这里所说的存带和读带是对ASCII 字符格式而言的，和以前所说对目标码格式的存带，读带操作不同。

(1) 存带操作如下：

当然这是在已编辑好一个文本之后才需要存带的。仍以小数二——十转换程序 PRO1 为例，假定它已经经过编辑存放在文本缓存区中了，现在需要把它存到磁带上。

| T | 重入编辑程序

；PRO1 显示第一行

| L |

/ · 此处或填该文本所包括的行数

OUT = T　　F = PRO1 ‹CR›　　T = 1 ‹CR›　　× ×

标记方框处为程序员应按键的地方，注意该文件名不得超过五个字符。

这样操作之后，AIM-65就立即开始转贮过程了，在转贮过程中，× ×处显示不断更换的00，01，02…字样，转贮完毕，AIM-65显示END字样。

对于应当注意使录音机操作和 AIM-65操作互相配合好的问题，前面已经交待过了，此处当然仍需同样注意，不必多说。

(2) 读带操作，这是把已存到带上的一个ASCII 码文件再读到内存，操作如下：

E

EDITOR

FROM = × × × × ‹CR›　　TO = × × × × ‹CR›

IN = T　　F = PRO1 ‹CR›　　T = 1 ‹CR›

LOAD　　F = PRO1　　BLK = × ×

BLK = 后的× ×处将不断更换显示00，01，02…

读带完毕，AIM-65显示END字样。

然后再用编辑程序中的 T 命令、L 命令，可把刚读入内存的PRO1程序全部文本内容打印出来。

5. R命令使用举例

前面说过 R命令的操作内容是从输入设备读若干行内容进文本缓存区，此处我们是把已存在磁带上的一段ASCII 字符文件读进正在进行编辑的文本缓存区中，为了简单起见，我们举以下的

例子进行R命令操作说明：

（1）假定一段以TEXT1命名的文本已用ASCII码格式存入带中，它的内容是：

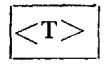
WE ARE LEARNING AIM-
65'S EDITOR.
IT IS INTERESTING.

这里不是源程序，只不过是两句英语。

（2）在AIM-65上再用E命令编辑一个文件如下，它的内容也只不过是两句英语。

<E>
EDITOR
FROM = 0200 TO = 02FF
IN = ‹CR›
TODAY IS MONDAY.
IT IS THE NINETEENTH
OF JULY.

（3）现在我们准备把磁带上的那两句英语插到正在编辑的这两句英语前面，就需要使用R命令，操作如下：



TODAY IS MONDAY
= <R>
IN = T　　F = TEXT1　　T = 1‹CR›

此处的T 命令是把行指针移到文本顶部，于是显示出正在编辑的文本第一行内容，TODAY IS MONDAY。R 命令是要把磁带上内容读到TODAY IS MONDAY 这一行之前去，按R键之后的操作和由带读 ASCII 码格式文本进内存操作相同， 读带完毕

AIM-65显示下一行内容，现在我们再用T和 L命令就可打印出编辑出来的最后文本如下：

WE ARE LEARNING AIM-

65'S EDITOR.

IT IS INTERESTING.

TODAY IS MONDAY;

IT IS THE NINETEENTH

OF JULY.

这里需要说明的是R命令只能把带上内容插入到正在编辑的文本某行的前面去，因此在按 R 键读带之前必须使用T命令和行指针位置调整命令找到待插入位置，然后才能使用R命令。

可见R命令对两个文本的合并是很有用的。

我们使用 R命令时亦可不是从磁带插入一段内容，而仍然从键盘插入一段内容，那么在按下R命令后，回答IN = 时用‹CR›回答而不是用T回答即可。这时R命令的操作功能类似 I命令，区别仅在于I命令只能插入一行内容，而 R 命令则可以插入若干行内容。

关于文本编辑命令的操作说明至此告一段落，下面我们再介绍汇编程序操作命令的使用方法。

三、汇辑程序操作命令

文本编辑程序只能把程序员用键输入的源程序放到了内存文本缓存区中，但是计算机并不能直接执行源程序，计算机只能直接执行目标程序，因此还必须把已存的源程序翻译成为目标程序，这个过程称之为汇编过程，完成这个任务的是汇编程序。

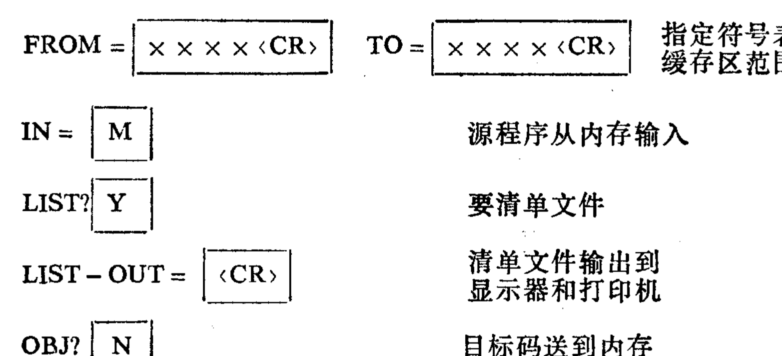
1. 汇编程序命令操作说明举例

我们仍以 PRO1程序为例，它已被放在文本缓存区中了，现在要使用汇编程序命令把它汇编成目标程序。

操作如下：

N　　　　　　　　这是汇编命令

ASSEMBLER



这样操作之后，AIM-65就开始汇编过程，第二遍扫描开始之后就打印出清单文件，我们把这个过程列在下面：

```
<N>
ASSEMBLER
FROM = 0600   TO = 0650
IN = M
LIST? Y
LIST – OUT = ‹CR›
OBJ ? N
PASS 1
PASS 2
 = = 0000
; PRO1
 = = 0000  BIN = $0300
; PRIMARY
; DATA  ADDRESS
```

```
= =0000  BCD = $0301
; RESULT
; ADDRESS
= =0000  MU10 = $0240
; *10 ROU-
; TINE
= =0000
                  * = $0200
= =0200
A000              LDY#00
AD0003            LDA  BIN
8501              STA  $01
=0207             NEXT
204002            JSR  MU10
990103            STA  BCD, Y
A501              LDA  $01
C8                INY
C003              CPY  #03
D0F3              BNE  NEXT
00                BRK
= =0215
                  * = $0240
= =0240           MU10
A200              LDX  #00
8600              STX  $00
A209              LDX  #09
D8                CLD
= =0247           LOOP1
```

```
18              CLC
6501            ADC  $01
9002            BCC LOOP2
E600            INC  $00
==024E          LOOP2
CA              DEX
D0F6            BNE LOOP1
8501            STA  $01
A500            LDA  $00
60              RTS
                RTS
  ERRORS=0000
```

为了弄清楚这个操作过程，我们对以下问题加以说明：

(1) 符号表缓存区，在 FROM=×××× TO=××××中规定的内存缓存区是用来存放源程序中使用的符号，如 PRO1程序中的 NEXT，BIN，BCD，MU10 等标号和符号地址，及其对应的地址单元都要由汇编程序把它们放在这个缓存区中。

要注意的是汇编过程总是跟在编辑过程之后进行的，那么符号缓存区和文本缓存区一般来说不能重叠，否则一经汇编之后就会将文本缓存区内容冲掉，那么源程序也就从内存中丢失了。

符号缓存区范围的大小由程序员视源程序中符号数目的多少大概估计而定。

(2) 两遍扫描：第一遍扫描(PASS1)的任务是在程序员指定的符号表缓存区形成符号表。第二遍扫描的任务是把源程序翻译成目标程序，并且检查有无语法错误，同时打印输出。

(3) 清单文件和目标文件

清单文件是经过汇编程序处理而得到的既包含符号指令又包含机器指令的文件，称为清单文件。

而目标文件则是经过汇编程序处理而得到的只包含机器指令的文件，称为目标文件。

我们观察一下上面列出的 PRO1 的清单文件可以看到，左半部是机器指令，右半部是符号指令，中间在迁标号地址处还插有对标号实际地址的说明。

对于 PRO1 我们上面只打印了清单文件而未打印目标文件，如果要求打印目标文件，则只要把编辑操作过程中 OBJ？后的回答改为Y即可得到。

源程序经过汇编程序处理之后就得到了计算机可以直接执行的目标程序。例如现在对 PRO1 程序由于已经经过汇编，我们就可以使 * = 0200 并用G命令使该程序执行。我们还可以把经过汇编得到的目标程序存到磁带上去，那么将来再执行 PRO1 程序时，就直接把它的目标程序由带送回到内存就行了。

现在我们重复说明一下汇编操作中对AIM-65提出的各问题，程序员应如何回答：

1）跟在FROM = 和 TO = 后面应回答符号缓存区的下限和上限地址。

2）跟在 IN = 后面应回答存放源程序的输入设备是什么，由于我们此处只介绍录音机使用的手控方式，而未介绍遥控方式，所以在 IN = 后面只能回答M，意思是源程序存放在内存中。只有录音机使用遥控方式时，此处才允许回答T，意思是源程序存放在带上。那么汇编时要在汇编过程中再从带上调进源程序，这点我们就不做介绍了。

对于录音机只接成手控方式时，若是源程序还在带上尚未读到内存的情况，必须在汇编操作之前先用读带命令（ASCII格式读带）把源程序调进内存，然后才能使用N命令对源程序进行汇编。

3）LIST？后面的回答可有Y和N两种，

回答Y表示要列出清单文件，回答N表示不要列出整个清单文件，只要列出语法检查出错情况。

4）LIST-OUT？后应回答清单文件或是语检结果输出到什么设备。可回答<CR>，意思是清单文件同时送到显示器和打印机，亦可回答P意思是清单文件送到打印机。

5）OBJ？可回答N亦可回答 Y。回答N表示汇编后形成的目标码送内存，不送其它输出设备，回答Y表示汇编后形成的目标码要送输出设备（打印机P或磁带 T），而不送内存。

6）OBJ-OUT＝这个问题只在OBJ？后回答Y时 AIM-65 才提出，可回答<CR>亦可回答P，亦可回答T。

现在我们把对 PRO1 汇编操作中需要将目标程序送到打印机的操作过程列在下面，供大家参考。

```
<N>
ASSEMBLER
FROM = 0600  TO = 0650
IN = M
LIST? N
LIST-OUT = P
OBJ? Y
OBJ-OUT = P
J
PASS 1
PASS 2
; 150200A000AD0003850
1204002990103A501C8C
003D0F30006E0
J ERRORS = 0000
```

; 160240A2008600A209D

81865019002E600CAD0F

68501A500600914

J; 0000030003

这里的；表示一块目标码的开始，跟在；后面的第一个字节是本块目标码的字节数，第二、三字节是本块目标码的开始地址，然后就是本块目标码的内容，最后两个字节则是本块目标码的检验和。PRO1汇编后得到的目标码内容分成了两块，第一块在第一个；后，共$(15)_H$个字节，起始地址是0200，检验和是06E0，第二块在第二个；后，共$(16)_H$个字节，起始地址是0240，检验和是0914。

至于第三个；后的内容按规定格式，第一个字节是00，第二个及第三个字节的内容是目标码所包括的数据块数（连本块在内）此例中连本块在内共计三个数据块，所以是0003，最后两个字节是本块检验和。

2. 汇编后形成的目标程序如何转贮到磁带

这有两种方法，第一种方法是在汇编过程完全结束之后，再用前面介绍过的目标码转贮磁带命令D (DUMP COMMAND) 来完成，第二种方法则是在汇编过程之中来完成，我们现在介绍第二种方法的操作步骤：

<N>

ASSEMBLER

FROM = 0600　TO = 0650

IN = M

LIST? N

LIST-OUT = P

OBJ? Y

OBJ-OUT = T　F = OBJ1　T = 1

```
PASS 1
PASS 2
ERRORS = 0000
```

这里在汇编操作中我们没有要求列出清单文件，由于语法完全正确，所以也未印出语检出错情况。

对于目标程序我们此处是要求送到磁带，所以还要回答文件名和录音机台号，这里的文件名是指要送到带上存起来的目标程序名，我们起名为OBJ1，汇编和目标程序存带过程结束之后AIM-65自动回监控程序，至于已存到磁带上的目标程序如何再读进内存这在前面已经做过说明，用L命令（LOAD MEMORY）即可，此处不再重复。

3．语检错误分类

若是在编辑过程中形成的源程序有语法错误，那么对这个有语法错误的源程序进行汇编的过程中，汇编程序将指出错误所在行，并指出是什么错误，最后还将用ERRORS来指出总的出错个数。

举例如下：若是源程序由于程序员的疏忽在编辑中送入文本缓存区时成为下面的样子，可以注意到这里漏掉了BCD = $0301这一行，另外STX $00错成为 STX #00那么把这个有语法错误的源程序进行汇编时，AIM-65将如何指出这两处错误呢？

```
BIN = $0300  ; PRIMARY
             ; DATA ADDRESS
MU10 = $0240; RESULT
             ; ADDRESS
* = $0200
LDY #00
LDA BIN
STA $01
```

```
NEXT JSR MU10
STA BCD, Y
LDA $01
INY
CPY #03
BNE NEXT
BRK
*=$0240
MU10 LDX #00
STX #00
LDX #09
CLD
LOOP1 CLC
ADC $01
BCC LOOP2
INC $00
LOOP2 DEX
BNE LOOP1
STA $01
LDA $00
RTS
```

下面我们再看汇编过程：

```
<N>
ASSEMBLER
FROM=0600 TO=0650
IN=M
LIST? Y
LIST-OUT=P
```

```
OBJ? N
PASS     1
PASS     2
= =0000        BIN= $0300
; PRIMARY
; DATA ADDRESS
= =0000        MU10= $0240
; RESULT
; ADDRESS
= =0000
                   * = $0200
= =0200
A000               LDY #00
AD0003             LDA BIN
8501               STA $01
= =0207            NEXT
204002             JSR MU10
99D37F             STA BCD, Y
* *ERROR           01
A501               LDA $01
C8                 INY
C003               CPY #03
D0F3               BNE NEXT
00                 BRK
= =0215
                   * = $0240
= =0240            MU10
A200               LDX #00
```

```
868878          STX #00
**ERROR         18
A209            LDX #09
D8              CLD
==0248          LOOP1
18              CLC
6501            ADC $01
9002            BCC LOOP2
E600            INC $00
==024F          LOOP2
CA              DEX
D0F6            BNE LOOP1
8501            STA $01
A500            LDA $00
60              RTS
                RTS
ERRORS=0002
```

可以看到清单文件中在STA BCD，Y 这一行下面标出ERROR01字样，指出此行有01类型错误。在STX#00这一行下面标出ERROR 18字样，指出此行有18类型错误。

最后一行ERRORS=0002指出共有二个语法错误。根据AIM-65的语检错误分类规定，01类错误是<未定义符号>，因为前面漏掉了BCD=＄0301这一行，所以BCD就成了未定义符号，程序员由 AIM-65指出的01类错误就会检查出这一点而把它补上。

18类错误是<非法操作数>程序员由此受到启发，再仔细查对一下STX #00这条指令发现STX指令是不允许立即数地址的，从而就知道应把 STX #00改成符合语法规定的 STX ＄00，然后再把修改后的源程序再行汇编，直到ERRORS=0000为止，这

时，汇编得到的目标程序才是可用的，才能交给AIM-65去执行。

我们这里只是通过一个例子告诉大家，AIM-65汇编程序是如何指出语法错误的，此例中只列出了二个错误的类型，至于其它各类语法错误我们就不再一一举例了，请大家查表。

下面我们列出语法错误分类表

SYM TBL OVERFLOW

这是符号表缓存区范围小了，不够用而产生溢出

01 未定义符号

02 重复定义的符号

03 非法操作码

04 无效的地址

05 不许可用累加器格式

06 应使用0页地址

08 标号未由字母开始

09 标号超过六个字符

10 标号或操作码中含有非字母非数字的符号

11 等值语句中有尚未定义的符号

12 无效的变址——必须是X或Y变址

13 无效的表达式

14 未定义的汇编伪指令

15 无效的零页地址

17 相对转移超出了范围

18 非法操作数

19 间址超出范围

20 A、X、Y、S和P不许可用作标号

21 程序计数器为负，此时汇编程序使之复位为0值。

四、用户功能键F_1、F_2和F_3

上面讲述的监控程序、编辑程序及汇编程序各种操作命令已

足够供我们在AIM-65上调试和运行源程序。为了扩大机器功能，方便用户，AIM-65还另设有三个用户功能键F_1、F_2和F_3它们的功能如下：

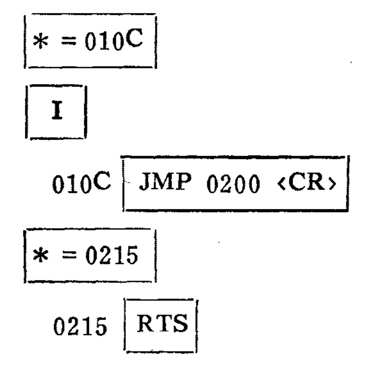
如果程序员把某段已输入AIM-65的程序的启动地址用一条JMP指令存放到F_1或F_2或F_3功能键所对应的规定地址单元中去，那么执行这段程序时只要按下F_1或F_2或F_3键即可使程序运行，而不必象已前所说的那样，要用＊和G命令。

F_1键对应的地址是＄010C，按下F_1键时，AIM-65显示的提示符是〕

F_2键对应的地址是＄010F，按下F_2键时AIM-65显示的提示符是〕

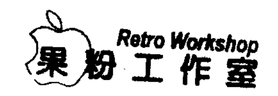
E_3键对应的地址是＄0112，按下F_3键时，AIM-65显示的提示符是Λ。

下面我们举例说明，仍以PRO1程序为例，假定它的目标程序已存放在内存中了，它的启动地址是＄0200，如果我们选定使用F_1功能键那么事先要从F_1键对应的地址＄010C开始，放进一条JMP＄0200的指令，并且在PRO1对应的目标程序结束处补上一条RTS指令，然后按下F_1键此时AIM-65就会自动从＄0200开始执行这个程序，并在遇到RTS指令时停止运行，返回监控程序控制之下，我们可将过程列出如下：



这样操作之后，再按F_1键，AIM-65就立即开始执行程序。

F_2和F_3键的用途同上，只是对应的地址应由＄010C改为＄010F和＄0112。



第四章　6502的外围接口芯片

美国ROCKWELL 公司生产的6500系列芯片中除了有象6502这样广泛使用的MPU之外，还有许多外围接口片子，其中以6522、6532、6520、6530使用最为广泛。而且 MOTOROLA公司6800系列外围接口芯片中的大多数，如6850、6820等也可以与6502方便地连用。本章的目的是较为系统地介绍这些芯片。在具体介绍这些外围芯片以前，我们先对其中的一些名词做简单说明。

并行输入输出（PIO）芯片　（如图4—1）

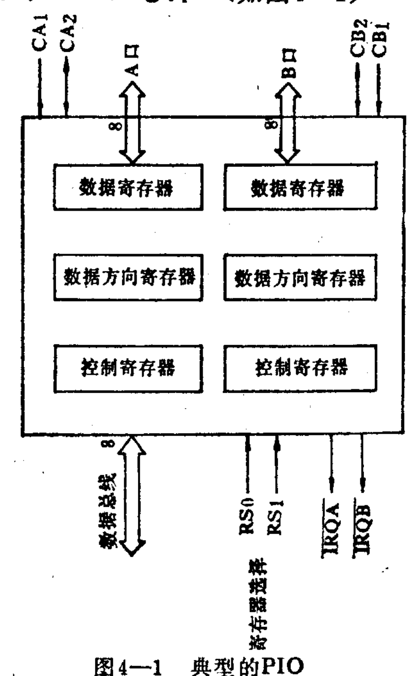


图4—1　典型的PIO

（图中箭头代表信息流动方向，以下同此不另注明）

PIO芯片一般是指至少可以提供两个 8bit 并行口的大规模集成电路接口片子，其中两个口的每一条线往往是可以由编程来控制其方向的。我们通常把控制并行口每一条线数据传送方向的那个寄存器称为数据方向寄存器。典型的 PIO芯片的框图如图4—1所示。它是外部设备与MPU连接的交界部分故称作接口芯片。为了保证它与外设之间数据传送的可靠性，芯片的每个口通常还附有两条联络("握手")线。为了和MPU传送数据，PIO芯片有8根线与MPU的数据总线相联，同时每个口还附有中断请求线。为了实现对上述那些附加的线的控制，PIO内部还专设有控制寄存器。通过寄存器选择线用来对上述的那些寄存器进行选择。

定时器 (TIMER)

在许多实际应用中，常常需要产生确定的延时。它一般可以用循环程序来完成，这叫做软件延时。它的优点是不需要添加硬件，但它却占用了处理机的工作时间，这在复杂系统中是不能允许的，因此往往用硬件来完成确定的延时，这就是定时器。

定时器常由8bit或16bit的寄存器组成。当它用来产生确定的延时的时候，常用系统时钟作为它的计数脉冲（系统时钟周期为1μs)。如果需要N微秒的延时，可事先向定时器写入数值N,然后启动定时器。每过1微秒，N 将被减1，当减到零时，定时器将产生一个输出电平，并向MPU请求中断。

定时器也可以用来测量一个外部脉冲的宽度或两个外部脉冲之间的时间间隔。这时将它事先写入零,定时器将进行累加计数，每隔1微秒它的内容将加1。当被测对象的延时到达时，将会有一个标志位被置1，并向MPU请求中断,读定时器的内容就测定了时间间隔。

定时器还可以用来产生单脉冲或脉冲序列，这两者都可以通过对定时器的编程来实现。

通用异步接收发送器 (UART)

UART 的主要功能是完成并——串和串——并转换，这种转换是由内部的移位寄存器及有关的逻辑来完成。

§4—1 6520外设接口适配器（PIA）

6520的主要功能是并行输入输出（PIO），它的引腿如图4—2，内部结构如图4—3。其引腿同MOTOROLA 6820完全相同。它主要用来做为外设（如键盘、显示器、打印机等）同6502MPU相连的接口芯片，故称它为外设接口适配器，也可称作PIO芯片。它是由+5V电源供电。

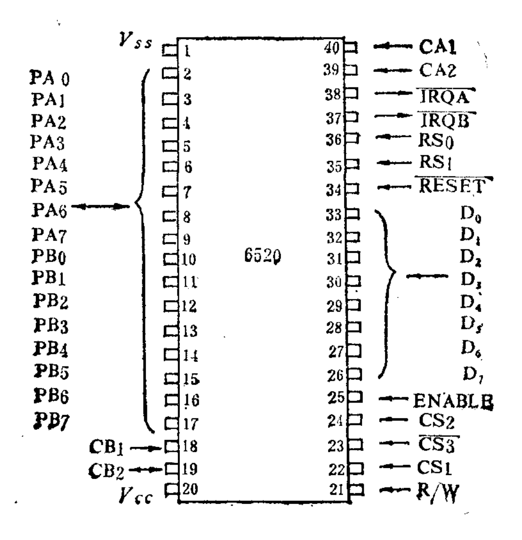


图4—2 芯片6520的引腿图

由图4—3可以看出，芯片包含两个输入输出口，即A口和B口。每个口8bit，都有一个缓冲器，每个口有一个输出寄存器，也就是说对于输出数据而言，A口、B口都是可以保存的。每个口的每根线究竟工作在输入还是输出的状况，那要看相应的数据方

向寄存器的状态。如果数据方向寄存器相应的位是0，则外设接口所对应的位就工作于输入状态；若数据方向寄存器相应的位是1，则外设接口所对应的位将工作在输出状态。数据方向寄存器的状态可以由编程来确定。但当芯片的复位($\overline{RESET}$)信号为0（低电平）时，所有寄存器的内容全为0，数据方向寄存器的内容也全为0。因此这将使两个口都工作在输入状态。这一点对于使用6520进

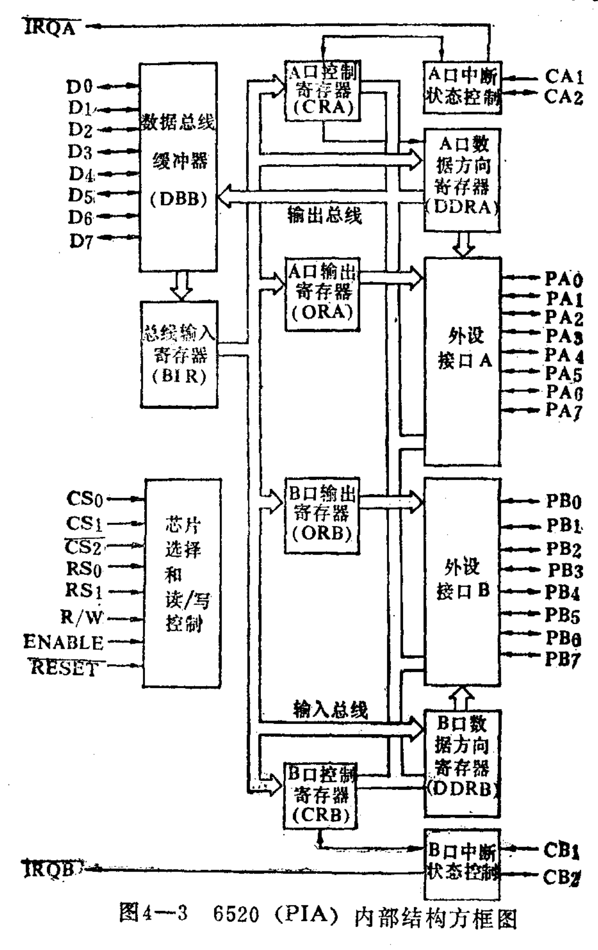


图4—3　6520（PIA）内部结构方框图

行某些机电设备的控制时很有好处，它可以保证在系统复位时不会由于输出电平而产生误动作。相应于A口和B口都有控制寄存器，它的功能我们将在下面详细讨论，它的内容不仅确定了各种各样的控制作用，而且它还包含了相应的状态信息。

由图4—3我们可以看到，每个口还设有两根外部控制线，即CA1、CA2和CB1、CB2。图中表明其中一根（CA1或CB1）是单方向的输入线，而另一根（CA2或CB2）则是双方向的输入输出线。它们的作用以及如何控制其方向，以下将会谈到。

正如图4—3所示，A口和B口在逻辑上是等效的，但是实际上这两个口在电气性能上是有差别的。图4—4A、B分别画出了两者的电原理图（只是缓冲器部分），从图可以看出：在输入方式下A口缓冲器由于接有一电阻，它相当于一个TTL负载；而B口则是高

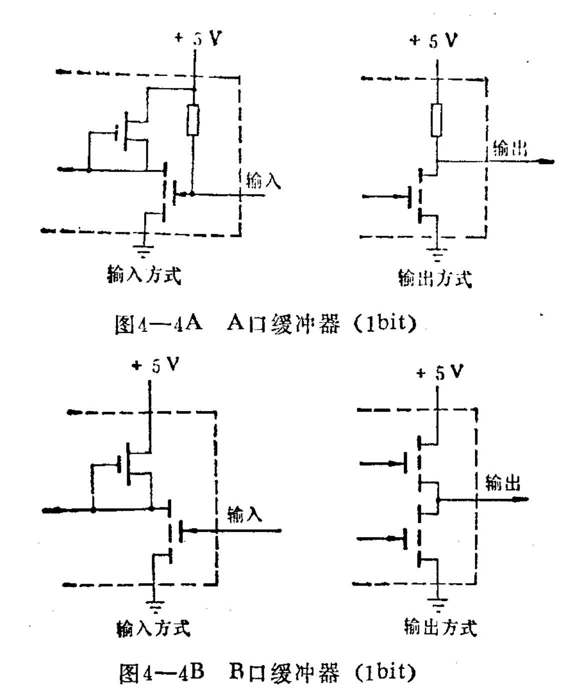


图4—4A　A口缓冲器（1bit）

图4—4B　B口缓冲器（1bit）

输入阻抗（大于1MΩ）。在输出工作方式，B口采用有源电路，而A口则是无源电路，因此B口的驱动能力高于A口。A口可以"灌"1.6mA，可以驱动一个标准的TTL负载。B口由于采用有源电路，在逻辑1时输出电压不会高于2.4V，但在高电平时仍可"拉"1mA，可用它来驱动达林顿晶体管开关。虽然高电平不大于2.4V，但仍能满足TTL电路的要求，但不能同CMOS电路直接相连。

现在我们再来看看图4—3的左边。数据总线缓冲器使6520内部的数据总线同系统的数据总线连接起来。图4—3中6520内部总线的所谓输入、输出是对外设接口缓冲器而言的。$\overline{IRQA}$和$\overline{IRQB}$分别是A口和B口的中断请求信号线。对芯片必须给定三根芯片选择线CS0、CS1和$\overline{CS2}$，这三根线原先MOTOROLA6820的设计者是为了在同时使用多个芯片的时候省去地址译码器，6520是仿照下来的。RS0、RS1是寄存器选择线。如上所述，我们已经介绍过两个控制寄存器、两个数据方向寄存器、两个输出寄存器以及两个外设接口缓冲器，一共八个寄存器。最后两个寄存器对于编程来说，可合称为输入输出寄存器（IORA和IORB)。即使是这样，两根寄存器选择线是不可能分别选择六个寄存器的，而且由于引腿条数的限制，不可能再增加寄存器选择线。为了解决这个问题，设计者只得安排让输入输出寄存器和数据方向寄存器使用同一个地址，而依靠内部的控制寄存器的 bit2的值来把它们区别开来。表4—1列出六个寄存器的地址分配。

由表4—1可以看出，A口或B口的控制寄存器可由RS0和RS1直接选择，而输入输出寄存器和数据方向寄存器除了RS0和RS1的值之外，还取决于相应的控制寄存器的 bit2值。因此在编程时首先必须对控制寄存器的内容予以确定，然后才能对其它的寄存器进行读写。

最后三根线是$\overline{RESET}$(复位)、R/W(读/写）和ENABLE(使能）线。复位线常和 6502 的复位线连在一起。读/写线是用来对

表4-1　　6520 的寄存器选择

RS_1	RS_0	A口控制寄存器bit2	B口控制寄存器bit2	所选中的寄存器
0	0	1	-	A口输入输出寄存器
0	0	0	-	A口数据方向寄存器
0	1	-	-	A口控制寄存器
1	0	-	1	B口输入输出寄存器
1	0	-	0	B口数据方向寄存器
1	1	-	-	B口控制寄存器

6520进行读写控制的，当它为高电平时是对芯片进行读操作，低电平时则进行写操作。R/W线是6520的输入线，它常和6502所输出的读写控制线——R/W线连在一起。使能线在使用中常与系统时钟$\phi 2$连在一起。

以下我们进一步对6520的控制寄存器进行讨论：

首先我们列出表4—2，说明控制寄存器各个bit的作用。对表4—2说明如下：

表4—2　　6520 控制寄存器

	7	6	5	4	3	2	1	0
A口控制寄存器CRA	A口中断请求1 $IRQA_1$	A口中断请求2 $IRQA_2$	CA_2控制			A口数据方向寄存器存取	CA_1控制	
	7	6	5	4	3	2	1	0
B口控制寄存器CRB	B口中断请求1 $IRQB_1$	B口中断请求2 $IRQB_2$	CB_2控制			B口数据方向寄存器存取	CB_1控制	

Bit7和Bit6都是A口（或B口）的中断请求状态信息位。前者由CA_1（或CB_1）的有效跳变置1，由对相应的外设接口缓冲器进

行读操作而清零，后者与前者相同，不同的只是它由CA2（或CB2）有效跳变置1，而不是CA1（或CB1）。

Bit5、4、3：CA2（或CB2）控制，下面详谈。

Bit2：0指示可对相应的数据方向寄存器进行读写。1则指示可对相应的输入输出寄存器进行读写。

Bit1.0：CA1（或CB1） 控制。下面我们列出表4－3予以说明。表4—3表明控制寄存器的Bit1是用来控制CA1（或CB1）所输入的中断信号究竟是正跳沿还是负跳沿有效；而Bit0则是控制

表4—3　　6520 的 CA1 或 CB1 控制

CRA—1（或CRB—1）	CRA—0（或CRB—0）	CA1（或CB1）中断输入	CRA—7（或CRB—7）中断标志	向MPU6502的中断请求 $\overline{IRQA}$（或$\overline{IRQB}$）
0	0	负跳沿有效	CA1（或CB1）的负跳沿将它置1	禁止中断请求——$\overline{IRQ}$线保持1电平
0	1	负跳沿有效	CA1（或CB1）的负跳沿将它置1	当中断标志CRA—7（或CRB—7）被置1时，$\overline{IRQ}$线变0
1	0	正跳沿有效	CA1（或CB1）的正跳沿将它置1	禁止中断请求——$\overline{IRQ}$线保持1电平
1	1	正跳沿有效	CA1（或CB1）的正跳沿将它置1	当中断标志CRA—7（或CRB—7）被置1时，$\overline{IRQ}$线变0

当中断标志位Bit7被置1时能否使向MPU6502的中断请求线$\overline{IRQA}$（或$\overline{IRQB}$）变成0电平，即控制能否向6502提出中断请求。当控制寄存器的Bit0是0时，即使有中断输入信号使中断标志位——即控制寄存器的Bit7——被置1，也不会向6502 MPU提出中断请求。$\overline{IRQA}$（或$\overline{IRQB}$）继续保持1电平，只有当控制寄存器的Bit0写入1以后，才能对MPU6502发出中断请求信号——$\overline{IRQA}$（或$\overline{IRQB}$）线变0电平。控制寄存器的中断标志CRA-7（或CRB-7）被置1以后，可通过对相应的外设接口缓冲器进行读操作来清除它。

以下我们连续列出三张表来说明控制寄存器的bit5、4、3是如何控制CA2（或CB2）的。

表4—4　　控制CA2（CB2）用作中断输入

CR A—5（或CRB—5）	CR A—4（或CRB—4）	CR A—3（或CRB—3）	CA2（或CB2）中断输入	CRA—6（或CRB—6）中断标志	向MPU6502的中断请求$\overline{IRQA}$（或$\overline{IRQB}$）
0	0	0	负跳沿有效	CA2（或CB2）的负跳沿将它置1	禁止中断请求——$\overline{IRQA}$（或$\overline{IRQB}$）保持1电平
0	0	1	负跳沿有效	CA2（或CB2）的负跳沿将它置1	当中断标志位——CRA—6（或CRB—6）被置1时$\overline{IRQA}$（或$\overline{IRQB}$）线变0电平
0	1	0	正跳沿有效	CA2（或CB2）的正跳沿将它置1	禁止中断——$\overline{IRQA}$（或$\overline{IRQB}$）线保持1电平
0	1	1	正跳沿有效	CA2（或CB2）的正跳沿将它置1	当中断标志位——CRA—6（CRB—6）被置1时，$\overline{IRQA}$（或$\overline{IRQB}$）线变0电平

表4—4表示当我们使控制寄存器的Bit5为0时，就可以使CA2（CB2）成为中断输入引腿。Bit4的值是控制中断输入的有效边沿是正跳还是负跳，如果为0——负跳沿有效；如果为1——正跳沿有效。这里所谓“有效”是指相应的跳变边沿可以使控制寄存器中的另一个中断标志位CRA-6（或CRB-6）置1。而Bit3的值是表明，当中断标志位（CRA-6或CRB-6）被置1时，能否向MPU提出申请中断——即使$\overline{IRQA}$（$\overline{IRQB}$）引腿输出为0电平。$\overline{IRQA}$（$\overline{IRQB}$）平时总是保持1电平的。当Bit3为0时即禁止中断，总是保持$\overline{IRQA}$（$\overline{IRQB}$）引腿为1电平；当Bit3的内容为1时，当中断标志CRA-6（CRB-6）被置1时相应的$\overline{IRQA}$（$\overline{IRQB}$）也相应地变为0电平——即向MPU6502提出申请中断。当中断标志位被置1以后，由MPU对相应的输入输出寄存器进行一次读，即可将

其复位为0。

表4—5和表4—6列出了控制寄存器的Bit5、4、3三位的内容可以对CB2(CA2)进行控制的各种功能。显然当Bit5为1时，CB2(CA2)即可做为输出信号使用。如前所述B口输出的驱动能力优于A口，所以这里对CB2和CA2的控制也略有不同。CB2是作为写

表4—5　控制CB2作为输出信号

CRB-5	CRB-4	CRB-3	方　式	说　明
1	0	0	作写操作"握手"信号	当MPU对6520B口的输入输出寄存器进行写操作时，将使CB2置0电平；而CB1的有效边沿则可使CB2恢复成1电平
1	0	1	脉冲输出	MPU对B口的输入输出寄存器写数据之后，CB2将保持1个时钟周期0电平
1	1	0	人工输出	置CB2为0电平
1	1	1	人工输出	置CB2为1电平

表4—6　控制CA2作为输出信号

CRA-5	CRA-4	CRA-3	方　式	说　明
1	0	0	作读操作"握手"信号	MPU对6520的A口输入输出寄存器进行读操作时，将使CA2置0电平；而CA1的有效边沿则可使CA2恢复为1电平
1	0	1	脉冲输出	MPU对A口的输入输出寄存器读数据之后，CA2将保持1个时钟周期0电平
1	1	0	人工输出	置CA2为0电平
1	1	1	人工输出	置CA2为1电平

操作“握手”信号来设计的，CA2是读操作“握手”信号来设计的。所以用户使用中最好把B口作为具有“握手”信号的输出口，A 口作为具有“握手”信号的输入口。“握手”两个字很形象地描述了经过B口（A口）与外部设备交换数据的过程。例如，我们将数据写入B口缓冲器，CB2同时变0电平，这个信口即可通知外设将数据取走，当外设将数据取走之后，它应经过CB1向6502发出“通知”——数据已取走。外设内部应有一信号源，在取走数据之后发出一脉冲，它的有效边沿将使CRB-7中断标志位置1和使CB2恢复为1电平，为下一次再写数据作好准备。CRB-7置1时，如果中断是不被禁止的（即CRB-0为1），则6520将向MPU发出中断请求信号，通知它再写数据。我们仔细地回味一下这个过程，确实是很有点“握手”的味道。读“握手”也是类似的，不再赘述。

由表4—5和表4—6可以看出，CB2(CA2)作输出信号，不仅可以与CB1（CA1）合作，成为写（读）操作的“握手”信号，也可以单独地输出脉冲，作为信号源使用。脉冲输出方式CB2(CA2)当对B口(A口)进行写（读）操作一次，即可输出一个窄的（宽度为1个时钟周期，一般是1微秒）负脉冲。人工输出方式下可以用程序来控制CB2(CB1)输出脉冲的宽度与周期。

以下我们对6520的使用列举一些例子，仅供参考。这些例子可以加深你对前面所叙述过的内容的理解。在举例之前，我们先列出各寄存器的代表符号：

PIACR——6520PIA控制寄存器（A口或B口）

PIADDR——6520PIA数据方向寄存器（A口或B口）

PIADR——PIA6520输入输出寄存器（A口或B口）

例1 写一个初始化程序，使6520的某一个口成为一个简单的不带控制线的输入口。

```
LDA  #0
STA  PIACR  ；控制寄存器清零，以便对数据方向寄存
```

```
            器寻址
STA PIADDR；使所有的线均为输入线
LDA #$04  ；Bit2为1，以便对输入输出寄存器寻址
STA PIACR
```

例2 写一个初始化程序，使6520的某个口成为一个简单的不带控制线的输出口

```
LDA #0
STA PIACR  ；控制寄存器清零，以便对数据方向寄存
             器寻址
LDA #$FF   ；使所有的线均为输出线
STA PIADDR
LDA #$04   ；Bit2为1，以便对输入输出寄存器寻址
STA PIACR
```

例3 写一个程序，使6520的某个口成为一个输入口，并带一根输入控制线，以这根线信号的正跳沿指明"数据准备好"或"数据可用"。

```
LDA #0
STA PIACR  ；控制寄存器清零，以便对数据方向寄存
             器寻址
STA PIADDR；使所有的线均为输入线
LDA #$06   ；CA1(CB1)正跳有效，但禁止向MPU
             请求中断
STA PIACR
```

本程序使用CA1(CB1)作"数据准备好"线。当此线上的信号出现正跳沿时，将会使控制寄存器的Bit7置1。许多情况下的键盘编码可用6520的这种结构。

例4 写一个程序使6520的B口成为一个输出口，并用一根控制线输出一个窄的打入脉冲，以指明"数据准备好"或"输出准备好"。

```
LDA #0
STA PIACRB  ；B口控制寄存器清零，以便对数据方
               向寄存器进行寻址
LDA #$FF    ；使所有的线成为输出线
STA PIADDRB
LDA #$2C    ；$2C=00101100，使CB2在对B口写
               操作之后输出窄脉冲，Bit2 为1，
               以便对输入输出寄存器寻址
STA PIACRB
```

例5 写一个程序使6520的A口成为一个带有“握手”控制线的输入口。

```
LDA #0
STA PIACRA  ；A口控制寄存器清零，以便对A口数
               据方向寄存器寻址
STA PIADDRA；A口所有的线成为输入线
LDA #$24    ；$24=00100100,CA2与CA1成为“握
               手”线，但禁止中断请求。
STA PIACRA
```

这个初始化程序使6520的A口在进行一次读操作时自动完成“握手”。

当然还可以列出更多的例子。读者可自行思考，如何控制CA2(CB2)输出一定宽度、周期的脉冲等。

§4—2 通用接口适配器(VIA) 6522

6522是在6520的基础上经过改进发展而成的。它是由PIO、定时器、移位寄存器的有机组合，它除了有6520 PIO方面的全

部功能外，另外又增加了定时器以及完成并——串和串——并转换的移位寄存器，故称它作通用接口适配器。它的内部结构如图4—5(a)所示，引腿分配则如图4—5(b)所示，芯片由+5V电源供电。

首先让我们来看图4—5(a)。如图它有两个8 bit 的双向口，即A口和B口；每个口设置一个输入输出寄存器，图中记作 ORA和ORB，它们都附有一个数据方向寄存器，即DDRA和 DDRB。同6520一样数据方向寄存器控制着相应口的每一根线是工作在输出状态还是工作在输入状态。如果数据方向寄存器的某一位内容是0，则那个口相应的那一条线就工作在输入状态；反之数据方向寄存器内容是1，则相应的那一条线就工作在输出状态。究竟数据方向寄存器的内容是0还是1是由编程决定的。因此可以用程序来控制A口及B口的功能。在芯片复位时(即引腿$\overline{RESET}$为0时)所有的寄存器的内容均为0，这同6520是一样的。两个口在驱动能力方面仍同 6520一样，B 口优于A口，在输入状态它们都相当于一个

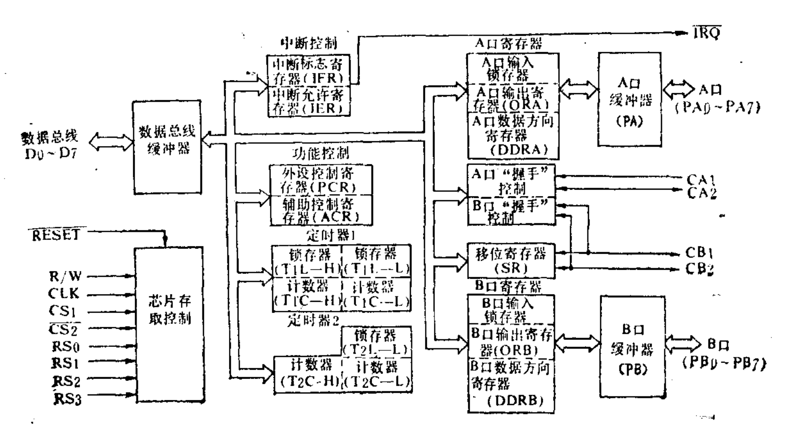


图4—5(a) 通用接口适配器 (VIA) 6522内部结构框图

TTL负载，另外不同于6520，两个口在做输入口使用时，都可以通过CA1 (CB1)的控制将输入数据锁存到内部寄存器。此外，A口和B口是可以带或不带“握手”信号的两个输入输出寄存器；而B

引脚	序号	6522	序号	引脚
V_{ss}	1		40	CA1
PA0	2		39	CA2
PA1	3		38	RS0
PA2	4		37	RS1
PA3	5		36	RS2
PA4	6		35	RS3
PA5	7		34	$\overline{RESET}$
PA6	8		33	D0
PA7	9		32	D1
PB0	10		31	D2
PB1	11		30	D3
PB2	12		29	D4
PB3	13		28	D5
PB4	14		27	D6
PB5	15		26	D7
PB6	16		25	CLK
PB7	17		24	CS1
CB1	18		23	$\overline{CS2}$
CB2	19		22	R/W
V_{cc}	20		21	$\overline{IRQ}$

图4—5(b) 6522引腿分配

口的Bit7和Bit6(PB7和PB6)可以受定时器控制，而分别作它们的输出和输入。图4—5中的CA1、CA2和CB1、CB2分别是A口和B口的控制线，除了CA2(CB2) 可由编程控制之外，CB1也可由编程控制其输入输出的功能。通过编程，CA1、CA2可在读写两种状态下成为A口的“握手”信号线，而CB1、CB2仅在写状态可以是B口的“握手”信号线。$\overline{IRQ}$是中断请求线，对 6522 来讲它是一条输出线，低电平(0电平) 有效。如果中断允许寄存器的某一位为1，当中断标志寄存器相应的位被置1时，将使$\overline{IRQ}$ 变成0电平而向MPU 6502提出中断请求。

现在我们再来看看图4—5左边的那些引线。经过数据总线缓

冲器使6522的内部总线同系统的数据总线连接起来，它们的引腿是D0～D7。$\overline{RESET}$是复位线，一般它将同MPU的复位线连在一起，当它为0电平时，6522将被强迫复位，所有内部寄存器将为0，但定时器1和定时器2以及移位寄存器除外，虽然不清除它们，但$\overline{RESET}$信号仍会把它们通通关闭，如果要定时器和移位寄存器工作，必须重新启动。R/W(读/写)线控制对6522的读写操作，当它为1电平时，可以进行读操作，而为0电平时则可进行写操作。CLK是时钟信号线，一般情况下这根线将同系统时钟ϕ_2接在一起，使6522和MPU之间的数据交换仅在ϕ_2为1电平时进行，ϕ_2也为芯片内部定时器和移位寄存器提供基准时钟。芯片选择线CS1、$\overline{CS2}$保证只有在CS1为1，$\overline{CS2}$为0电平时才能对芯片进行存取。由图4—5可以看出6522芯片有四条寄存器选择线，即RS0、RS1、RS2和RS3。这四根线通常与MPU的地址线相连，以便MPU可以选择6522的十六个内部寄存器进行读写，表4—7列出了6522的十六个内部寄存器分配。

表4—7中有两个地址均是A口输入输出寄存器，即0001或1111。前者就是一般PIO芯片的输入输出口，并带有“握手”信号CA1、CA2；而后一个地址则是一个不带有“握手”信号的简单的8bit输入输出口。

关于图4—5(a)我们就介绍到这里。因为6522无论在功能的灵活性或是多样性方面都是很先进的，因此如果一下子将各种编程的方法全盘托出，反而会使你不易理解和接受，现在我们由简单到复杂逐步向你介绍，使你在本节结束的时候将对6522有较全面的了解。

首先我们来看看6522用做简单的输入输出的情况：在这种情况下先要对数据方向寄存器的内容进行写入，以控制A口或B口的数据传送方向。例如我们想要用A口作简单的8位输入口，而B口作简单的8位输出口。

表4－7　　　　　　6522内部寄存器分配

RS3	RS2	RS1	RS0	寄　存　器
0	0	0	0	B口输入输出寄存器(ORB)
0	0	0	1	A口输入输出寄存器(ORA)
0	0	1	0	B口数据方向寄存器(DDRB)
0	0	1	1	A口数据方向寄存器(DDRA)
* 0	1	0	0	定时器1计数器低八位(T1L－L/T1C－L)
** 0	1	0	1	定时器1计数器高八位(T1C－H)
0	1	1	0	定时器1锁存器低八位(T1L－L)
0	1	1	1	定时器1锁存器高八位(T1L－H)
1	0	0	0	定时器2计数器低八位(T2L－L/T1L－L)
1	0	0	1	定时器 2 计数器高八位(T 2 C－H)
1	0	1	0	移位寄存器(SR)
1	0	1	1	辅助控制寄存器(ACR)
1	1	0	0	外设控制寄存器(PCR)
1	1	0	1	中断标志寄存器(IFR)
1	1	1	0	中断允许寄存器 (IER)
1	1	1	1	A口输入输出寄存器(ORA)但“握手”无效

* 注：这个地址对定时器 1 计数器低八位只能进行读操作，而写操作在这个地址上是对定时器 1 的锁存器低八位。

** 注：在这个地址进行写操作时，将把定时器 1 低八位锁存器的值自动传送到低八位计数器，并同时启动计数器开始计数。

```
LDA  #$FF    ；每bit都是1，使所有的线为输出线
```

```
    STA   DDRB
    LDA   #0        ；每 bit 都是0，使所有的线为输入线
    STA   DDRA
```

如果我们让二进制数 **01101001**（等于＄69)输出到B口，则：

```
    LDA   #$69
    STA   ORB
```

如果我们要将A 口的输入数据读进来，然后存放在存贮器某个单元(LOC1)：

```
    LDA   ORA
    STA   LOC1
```

以上这种简单的输入输出只在少数情况下使用。实际情况是往往在读写A口和 B 口之前，需要检查一个由外设送出的状态信号。这个状态信号告诉MPU数据已经准备好或可以发送，而且在数据已由 MPU 取走或发送时，MPU送出一个状态信号通知外设数据已取走或数据已发送。这一对状态信号(有时也称控制信号)就叫做“握手”信号。6522的 A 口B口都设有两根线，通过编程这两对线可以提供两个口在与外设交换数据时的“握手”信号。但注意A口的CA1、CA2 可以成为输入或输出的“握手”线，但CB1、CB2只能成为B口输出时的“握手”线。以下我们就来介绍对6522的两对控制线的控制。

6522内部外设控制寄存器（PCR，地址是1100）是为了控制CA1、CA2和CB1、CB2而设计的，它的组成如下：

表4—8　　外设控制寄存器 PCR

位	7	6	5	4	3	2	1	0
功能	CB2 控 制			CB1控制	CA2 控 制			CA1控制

由表4—8可以看出PCR—4和PCR—0分别控制CB1和 CA1，

而PCR的Bit5～7和Bit1～3则分别控制CB2和CA2。如果PCR-0的内容为1，引线CA1上输入的信号为正跳变将会把相应的中断标志位置1，反之如果PCR-0的内容为0，负跳变将使中断标志置1（中断标志寄存器见表4—9）。如果CB1作B口的控制线使用时，PCR-4对它的控制与PCR-0对CA1的控制完全相同。即PCR-4内容为1，引线CB1上输入的信号正跳变会把相应的中断标志位置1，反之如果为0，则负跳变将把中断标志置1，读或写A口（B口）将会把中断标志清0。但正如图4—5所示，CB1还可以作为移位寄存器的输入输出线使用。如果移位寄存器被使用，(由辅助控制寄存器的内容决定）CB1将为移位寄存器提供时钟信号的输入或输出。这一点以下还将详谈。

表4—9列出了中断标志寄存器，由表我们可以看出CA1、CA2、CB1、CB2、SR、T_2、T_1一共七个中断源各自都有它们自己的标志位。当相应的条件满足时，其标志位将会被置1。如果其对应的位允许中断时（见表4—17 中断允许寄存器)将会向MPU申请中断。七个中断源只要有一个被允许申请中断，则表4—9所列出的那个中断请求位(bit7）就将会是1。当有多片6522在同一系统中时，中断源可能来自好几片，为了确定中断源究竟来自哪一片，可以先查询IFR的Bit7。如果它为0，则可以肯定不是这一片，而立即转到查询下一片，而不必将IFR的各个位都查一遍。因而可以缩短中断源查找时间。

表4—9　　中断标志寄存器(IFR)

位	7	6	5	4	3	2	1	0
功能	中断请求	T_1	T_2	CB1	CB2	SR	CA1	CA2

以下我们来谈谈外设控制寄存器(PCR)对CB2和CA2的控制。

CA2由PCR的Bit1～Bit3 控制。通过编程可以使这根引线作为一条中断输入线或是一条外设控制输出线。后一种情况CA2是一条输出线，它同CA1 配合可完成A口在读或写过程中的“握手”操作，同时还可以作为一根窄脉冲输出线，或人工控制的信号输

表4—10　　　PCR 对 CA2 的 控 制

PCR-3	PCR-2	PCR-1	方　　式
0	0	0	CA2负跳沿中断方式：CA2 输入信号的负跳沿将把中断标志CA2(IFR-0)置1。读或写A口输入输出寄存器将清除中断标志位(IFR-0)，或者对IFR-0写入一个1也将清除中断标志IFR-0
0	0	1	CA2负跳沿中断方式：CA2 输入信号的负跳沿将把中断标志CA2(IFR-0)置1。读或写A口输入输出寄存器不能清除中断标志(IFR-0)，对IFR-0写入一个1可清除IFR-0
0	1	0	CA2正跳沿中断方式：CA2 输入信号正跳沿将把中断标志CA2(IFR-0)置1。读或写A口输入输出寄存器可清除中断标志IFR-0
0	1	1	CA2正跳沿中断方式：CA2 输入信号正跳沿将把中断标志CA2(IFR-0)置1。对IFR-0写入1可清除IFR-0中断标志。
1	0	0	CA2“握手”输出方式：读或写 A口输入输出寄存器将使CA2引线输出为 0 电平，而引线CA1上输入信号的有效跳变将使它恢复为1电平。
1	0	1	CA2脉冲输出方式：在读或写A 口输入输出寄存器之后，CA2引腿上将保持一个时钟周期为0电平。
1	1	0	CA2输出低电平方式：在这种方式中CA2引腿将保持低(0)电平输出。
1	1	1	CA2输出高电平方式：在这种方式中CA2引腿将保持高(1)电平输出。

表4—11　　PCR 对 CB2 的控制

PCR-7	PCR-6	PCR-5	方式
0	0	0	CB2负跳沿中断方式：CB2 输入信号的负跳沿将把中断标志CB2(IFR—3)置1。读写B口输入输出寄存器将清除中断标志CB2(IFR-3)；或者对IFR-3 写入1也可以清除IFR-3。
0	0	1	CB2负跳沿中断方式：CB2输入信号的负跳沿将把中断标志CB2(IFR-3)置1。对IFR-3写入1可以清除IFR-3；读或写B口输入输出寄存器不能清除中断标志CB2。
0	1	0	CB2正跳沿中断方式：CB2 输入信号的正跳沿将把中断标志CB2(IFR-3)置1。读或写B口输入输出寄存器将清除中断标志CB2(IFR-3)；或者对IFR-3写入1,也可以清除IFR-3。
0	1	1	CB2正跳沿中断方式：CB2 输入信号的正跳沿将把中断标志CB2(IFR-3)置1。对IFR-3写入1可以清除IFR-3。
1	0	0	CB2“握手”输出方式：写B口输入输出寄存器将置引腿CB2为低(0)电平。引线CB1的输入信号将使它恢复成高(1)电平。
1	0	1	CB2脉冲输出方式：在写B口输入输出寄存器之后，CB2引线上将保持一个时钟周期的0电平。
1	1	0	CB2输出低电平方式：在这种方式中，CB2引腿将保持低(0)电平输出。
1	1	1	CB2输出高电平方式：在这种方式中CB2引腿将保持高(1)电平输出。

出线。而在前一种情况，CA2是一条输入线，由于区别正跳或负跳而置相应的中断标志，以及区别清除中断的办法，而出现了四种不同的情况(见表4—10)。

如果 6522 中的移位寄存器禁止使用（由辅助控制寄存器的

Bit2～Bit4决定)，则外设控制寄存器的Bit5～Bit7 对引腿CB2的控制与上面谈的对CA2的控制非常相似。详见表4-11。

从表 4-10 和 4-11 可以看出 PCR-3～PCR-1和 PCR-7～PCR-5 的每一位控制着一种功能。如 PCR-3 (PCR-7) 即控制CA2(CB2) 是输入还是输出，在输入方式中PCR-2 (PCR-6) 即控制是正跳沿还是负跳沿有效，PCR-1 (PCR-5) 则控制清除中断标志的方法。

以下我们举一个例子来说明外设控制器的使用。如图 4-6 所示，我们假定从外设无论何时来一个正跳沿“数据准备好”信号

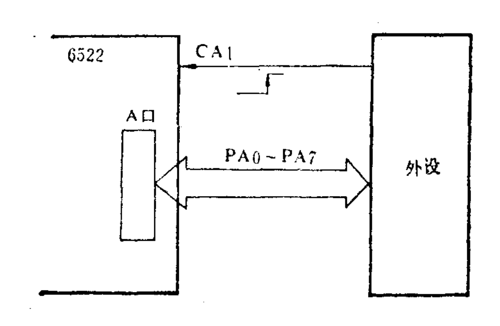


图4-6 “准备好”即读数据

以后，6522即采集数据，并将它存放到某个存贮单元。这个程序如下：

```
        LDA  #0       ；所有的线均是输入线
        STA  DDRA
        LDA  #$01
        STA  PCRA     ；CA1 正跳沿置中断标志IFR-1
                        为1
WAIT    LDA  IFR      ；读中断标志寄存器
        AND  #$02     ；$02=00000010，查询CA1中
                        断标志是否为1
```

```
BEQ  WAIT    ；如果为0，说明 CA1 引线上并
                未出现过正跳信号
LDA  ORA     ；读A口，收集数据
STA  LOC     ；存放到某个存贮单元
```

我们选6522 A口及 CA1 来完成这个任务。用上面的程序对6522A口进行编程，即可实现。

现在我们来介绍 6522 内部的辅助控制寄存器（ACR)。表4-12列出了辅助控制寄存器(ACR)的每一位的功能。由表可以看出Bit0和Bit1分别控制 A 口和 B 口的输入是否锁存。注意，6522 A口或B口的输入同输出是不对称的，输出总是被锁存的，所以人们常常把输入输出寄存器简称为输出寄存器。而输入是不是锁存则需要由辅助控制寄存器的Bit1和 Bit0 的内容所决定。如果是0则所相应的口是不锁存的；而内容是1则相应口的输入是被锁存在6522内部的输入锁存器中。更具体地说，是当CA1(CB1) 引腿上的有

表4—12　　　6522的辅助控制寄存器(ACR)

位	7	6	5	4	3	2	1	0
功能	定时器1控制		定时器2控制	移位寄存器控制			B口输入锁存允许	A口输入锁存允许

效跳变边沿信号将相应的中断标志置1时，就将这时出现在 A 口(B口）输入引线上的数据锁存到芯片内部，在此之后即使输入引线上的数据发生变化，对已锁存的数据也不会产生影响，这些数一直保存到CA1 (CB1)上接收到下一个脉冲为止。因此对于锁存的情况，读输入口所得到的值是锁存器的值，而不一定正是该口输入引线上的值，对于不锁存的情况，读输入口的值总是连到该口引线上的值。

下面我们来介绍6522内部的两个定时器，即T_1和T_2。T_1由两

个八位锁存器和一个十六位的计数器组成。用锁存器存放给计数器加载的数值，加载以后，计数器以系统时钟速率进行减量计数，计数到零时，中断标志（IFR-6）置1，如果允许则向MPU发出中断请求。表4—7中列出有关定时器1的四个地址。下面我们用表4-13和表4—14来说明对这四个地址写和读时的具体操作。

表4—13　　对 T_1 的写入

RS3	RS2	RS1	RS0	操　　作
0	1	0	0	写入低八位锁存器(T1L-L)
0	1	0	1	写入高八位锁存器(T1L-H)，写入高八位计数器(T1C-H)。送低八位锁存器到低八位计数器。T_1中断标志（IFR-6)清零。
0	1	1	0	写入低八位锁存器（T1L-L)
0	1	1	1	写入高八位锁存器（T1L-H)。T_1中断标志(IFR-6)清零。

表4—14　　对 T_1 的读出

RS3	RS2	RS1	RS0	操　　作
0	1	0	0	读T_1低八位计数器（T1C-L)，T_1中断标志(IFR-6)清零。
0	1	0	1	读T_1高八位计数器(T1C-H)
0	1	1	0	读T_1低八位锁存器(T1L-L)
0	1	1	1	读T_1高八位锁存器(T1L-H)

表4—15　　　　ACR控制T_1工作方式

ACR-7 输出允许	ACR-6 “连续运行”允许	方　　　式
0	0	单稳方式。禁止PB7输出。
0	1	连续运行方式。禁止PB7输出。
1	0	单稳方式。允许PB7输出。
1	1	连续运行方式。允许PB7输出。

注意，处理器不能直接对低八位计数器(T1C-L)进行写入，而是当处理器对高八位计数器（T1C-H）写入时，自动地将低八位锁存器(T1L-L)的内容加载到T1C-L。在写入锁存器时，不会影响正在计数中的减量计数器的值。

由表4-12所列出的辅助控制器（ACR）功能可以看到，ACR-7和ACR-6决定着定时器1(T_1)的工作方式，一共有四种选择，如表4-15所示。显然表中的四种选择实际上是两种工作方式：单稳和连续运行，通过 ACR-6 是0或是1来控制，ACR-7 是 0 或是1控制着B口的PB7引腿是否输出单稳或连续运行的信号。所谓单稳方式是指每次对T_1加载只能在减量计数器为零时，产生一次中断，而连续运行方式则将连续产生中断。前者如果允许在 PB7上将产生一个单脉冲——其脉冲宽度由编程决定；后者如果允许PB7上将产生方波。

例如：要求 PB7上输出一个300微秒宽的负脉冲，应如下对6522的T_1编程。

```
LDA   #$80   ; ACR-7=1  ACR-6=0  使T1为单稳
               方式，PB7允许输出
STA   ACR    ; 将使PB7输出为1
LDA   #$2C   ; (300)D = $12C。将低八位写入
               T1L-L
```

```
STA  T1L-I
LDA  #$01
STA  T1C-H;  写入 T1C-H，启动计数器减量计数，
             PB7为低电平。到计数器减到零PB7恢
             复为高电平。
BRK
```

由上面的程序可以看出首先必须对T_1的低八位锁存器加载，当写入高八位计数器时，会自动地把低八位锁存器的数值送到低八位计数器，也把高八位计数器的内容复制到高八位锁存器，T_1的中断标志被清除，同时启动计数器以系统时钟的速率开始减量计数，PB7 在写 T1C-H之后变成低电平。当计数器到零时，PB7变高电平，同时使T_1中断标志位置1，如果允许6522 的$\overline{IRQ}$引线将变成0电平，向MPU 提出中断请求，在此之后，计数器仍按系统时钟速率继续减量计数。这就使得用户可以用读计数器值的方法来计算中断产生以来的时间，但由于T_1中断标志已经置位成1，所以如果不再用一条写 T1C-H 指令来清除它，是不可能再置位的。

定时器1还可以工作在连续运行方式。它与前者不同的是：ACR－6＝1，如果允许PB7上输出的是方波，也就是说当启动计数器以后，将以系统时钟速率减量计数，每计数到零，PB7 的输出（如果允许）将倒相一次；同时自动将锁存器的内容再送入计数器，以此内容再重新开始减量计数，中断标志被置位。中断标志不一定用重写定时器来清除，可用读 T1C-L 或直接写中断标志的方法来清除。如将一个新的值写入锁存器将不会影响正在进行的减量计数，只能改变下一次计数输出的周期。而对高八位计数器的重新写入将使定时器1按新的数据工作；在计数器减到零之前，处理器连续重写高八位计数器将阻止计数到零。

例如使PB7输出半周期为1000微秒的方波，对T_1应如下编程：

```
LDA  #$C0
STA  ACR     ; ACR-7=1, ACR-6=1, 连续运
               行方式，允许PB7输出
LDA  #$D8
STA  T1L-L   ; (1000)D=$3D8
LDA  #$03
STA  T1C-H   ; 启动计数
BRK
```

定时器2(T_2)包括一个16bit减量计数器和一个 16bit 的锁存器。它只能工作在单稳方式，但却可以对B口引腿 PB6 上出现的负脉冲进行计数。辅助控制寄存器 ACR-5 位的内容可以控制究竟选择单稳方式或是计数方式。若ACR-5=0则工作在单稳方式；ACR-5=1则可对B口引腿 PB6 上出现的负脉冲进行计数。单稳方式的工作与T_1很类似，它所组成的16bit计数器可以按系统时钟速率进行减量计数。定时器2的寻址见表4—16。

表4—16　　T2 的 寻 址

RS3	RS2	RS1	RS0	写 操 作	读 操 作
1	0	0	0	写T2L-L (低八位锁存器)	读T2C-L(低八位计数器)清除中断标志(IFR-5)
1	0	0	1	写T2C-H (高八位计数器) 送T2L-L到T2C-L。清除中断标志	读T2C-H

显然定时器2高八位计数器是可读也可写的，而低八位锁存器是只写，低八位计数器是只读的。

例如利用T_2产生2048微秒的延时，可对T_2进行如下的编程：

```
LDA  #$0
STA  ACR     ; 置T2为单稳方式
```

```
        STA   T2L-L
        LDA   #$08     ; (2048)D = $0800
        STA   T2C-H    ; 启动T2
        LDA   #$20     ; $20 = (00100000)B
WAIT    BIT   IFR      ; A∧(IFR)，查询T2中断标志位是否
                       ; 为1
        BEQ   WAIT     ; 不为1则继续查询
        LDA   T2C-L    ; 清除中断标志
        BRK
```

由于T_2工作在单稳方式没有输出端，所以不能象T_1那样直接利用某个引腿输出单脉冲。但是我们仍然可以利用B口的某条引腿输出单脉冲，则程序可改为：

```
        LDA   #$FF
        STA   DDRB     ; B口所有线为输出线
        LDA   #0
        STA   ACR      ; 置T2为单稳方式
        STA   T2L-L
        LDA   #$08     ; (2048)D = $0800
        STA   T2C-H    ; 启动T2
        LDA   #$20
        STA   ORB      ; 将$20输出到B口
WAIT    BIT   IFR
        BEQ   WAIT
        EOR   #$FF     ; 将累加器A中的内容变成$DF，即
                       ; 每位都反相
        STA   ORB      ; 使B口每条输出线上的电压反相
        LDA   T2L-L    ; 清中断标记
        BRK
```

运行上面的程序，即可从B口的 PB5 上得到给定宽度的单个正脉冲。

T_2虽然没有连续运行方式，但是T_2却可以对B口引腿 PB6上负脉冲的预定个数进行计数。这首先是通过对T_2进行写入某个预定的值N，PB6上出现一个脉冲，即将T_2中的N减1。当减到零时，中断标志被置位。例如对B口引腿PB6上的脉冲计数：（假定计十个脉冲）

```
        LDA   #0
        STA   DDRB      ；B口所有线为输入线
        LDA   #$20
        STA   ACR       ；置T2为计数方式
        LDA   #$A       ；计十个脉冲
        STA   T2L-L
        LDA   #0
        STA   T2C-H     ；启动T2计数
        LDA   #$20
LOOP    BIT   IFR       ；查询T2中断标志为1?
        BEQ   LOOP      ；不为1继续查询
        LDA   T2L-L     ；清除T2中断标志
        BRK
```

以上我们曾多次谈到中断标志的清除和置位问题。现在我们

表4—17　　中断允许寄存器(IER)

位	7	6	5	4	3	2	1	0
功能	置1/清0	T_1	T_2	CB1	CB2	SR	CA1	CA2

来谈谈6522内部的中断允许寄存器(IER)，表4—17列出了它的分配情况。表中 Bit0～6 列出了每一位所对应的允许中断的信号和

对象。如果该位为1，则允许当中断标志寄存器(IFR)相应的位被置1时向MPU 提出中断请求——即$\overline{IRQ}$引腿变成0电平。利用中断允许寄存器的Bit7可以对每一个你所要求的位事先置1或清零。如果IER－7＝0，写IER将把为1的相应的位清零；如果IER－7＝1，将把为1的相应的置1。例：

```
LDA  #$7F   ; $7F=01111111
STA  IER    ; 将IER的Bit0～6清零
LDA  #$F0   ; $F0=11110000
STA  IER    ; 将IER的Bit4～6置1
```

以下我们来谈谈6522内部的移位寄存器。

6522内部的移位寄存器可以完成串——并转换和并——串转换。移位速率可由三个时钟源来提供：定时器2，系统时钟 ϕ_2 以及外部时钟源。串行数据的输出或输入是由引腿 CB2 完成，CB1 引腿可以引入外部时钟源或者输出内部产生的移位脉冲去控制外部装置中的移位操作。移位寄存器的工作由6522内部的辅助控制寄存器(ACR)的Bit2～4控制。Bit4 控制输出/输入选择。Bit4＝1移存器输出工作方式；Bit4＝0移存器输入工作方式。全部控制功能见表4—18。现说明如下：

000方式：用来禁止移位寄存器，此方式中MPU 可对移位寄存器进行读写。

001方式：此方式中移位速率由T_2 的低八位锁存器中的内容控制，引线 CB1上的移位脉冲可输出控制外部装置的移位操作。启动移位由执行对移存器的读出或写入开始。数据在移存器内部总是向左移动，即先进入移位寄存器的Bit0，然后向它的高位移动。八次移位累计完毕相应的中断标志IFR-2将被置1。

010方式：此方式中移位寄存器以系统时钟ϕ_2的速率移位(实际上是每2微秒移动一位)；CB1 可输出移位脉冲来控制外部装置移位操作。数据在移位寄存器中依然是向左移动，读或写移位

寄存器可启动移位。九个移位脉冲之后中断标志IFR—2将被置1，而且引腿CB1上的移位脉冲也就停止。

011方式：外部时钟控制下的移位输入。此方式中引腿CB1成为输入端，外部移位脉冲加在这条引线上可以控制移位。每输入八个移位脉冲中断标志 IFR—2 被置1。读写移位寄存器可启动移位并复位中断标志。数据在移位寄存器中依然是向左移动。为了进一步了解移位输入请看以下例子。

例在T_2控制下移入8bit数据，并将此数据存放在LOC1存贮单元。

```
        LDA   #0
        STA   ACR      ; 禁止移位寄存器
        LDA   #$04     ; $04=00000100
        STA   ACR      ; 置T2控制下移位输入方式，
                         置T2为单稳方式
        LDA   #$28     ; $28=(40)D
        STA   T2L—L    ; 延迟40微秒
        LDA   #0
        STA   T2C—H    ; 启动T2
        LDA   SR       ; 启动移位寄存器
        LDA   #$04
WAIT    BIT   IFR      ; 查询移存器中断标志
        BEQ   WAIT     ; 中断标志=0则继续查询。
        LDA   #0
        STA   ACR      ; 禁止移位寄存器
        LDA   SR       ; 取移入的数据
        STA   LOC1     ; 存入指定的存贮单元
        BRK
```

又例在系统时钟ϕ_2控制下移入8bit数据，（如上所述，实际上

这是每二个系统时钟移动一位。)并将它放在LOC2存贮单元。

```
        LDA   #0
        STA   ACR   ；禁止移位寄存器
        LDA   #$08 ；  $08 = 00001000
        STA   ACR   ；置φ2控制下移位输入方式
        LDX   #4
        LDA   SR    ；启动移位寄存器
WAIT   DEX
        BEQ   WAIT ；延时18个时钟周期
        STX   ACR   ；禁止移位寄存器
        LDA   SR    ；取数据
        STA   LOC2  ；存入指定存贮单元
```

现在来看移位寄存器的输出方式。只要使ACR－4＝1，6522内部的移位寄存器即工作在输出状态。首先将移位寄存器的第七位数据移到CB2引腿上，同时也移到第零位上，而第六位则移到

表4—18

ACR—4	ACR—3	ACR—2	方式
0	0	0	禁止移位寄存器
0	0	1	定时器2控制下的移位输入
0	1	0	ϕ_2脉冲控制下的移位输入（实际上每二个ϕ_2移动一位）
0	1	1	外部时钟脉冲控制下的移位输入
1	0	0	定时器2决定的速率连续运行输出
1	0	1	定时器2控制下移位输出
1	1	0	ϕ_2脉冲控制下的移位输出（实际上每二个ϕ_2移动一位）
1	1	1	外部时钟脉冲控制下的移位输出

第七位上，……依此类推。简言之，即循环左移，第七位与CB2相连。同输入一样CB1可以作为输出提供移位脉冲，也可以作为输入提供外部移位脉冲以实现SR的移位操作。通过ACR-2、ACR-3可以选择四种工作方式（见表4—18）。

100方式：这种方式中移位次数计数器停止工作，所以它一直连续循环移位，它与101方式非常类似，也是以T_2控制移位速率。

101方式：此方式T_2控制移位速率。每次读或写移位寄存器均将使移位次数计数器清零，并启动移位寄存器，八个移位脉冲将数据左移到CB2引腿上。与此同时，CB1引腿也输出八个移位脉冲。这八个移位脉冲之后自动禁止移位操作，而且将IFR-2置1。如果再度对移位寄存器进行读写将重新开始移位。

110方式：与101方式相同，只是此种方式是以系统时钟Φ_2控制移位(实际是每二个系统时钟移动一位)，而不是T_2控制移位。

111方式：此方式中，由外部装置向引腿CB1提供移位脉冲。每当移位次数计数器计满八个脉冲，即使中断标志IFR-2置1，但不禁止移位。读或写移位寄存器将使IFR-2清零，和移位次数计数器开始重新计数。以下几个例子可以帮助读者进一步了解6522移位寄存器的输出功能。

例 将存贮单元LOC3中的数据以T_2决定的速率连续移出8 bit。

```
LDA  #0
STA  ACR  ；禁止移位寄存器
LDA  #$10 ； $10=00010000
STA  ACR  ；置T2速率连续移位输出方式
LDA  #$80
STA  T2L—L；置T2
LDA  #0
STA  T2C—H；启动T2
```

```
LDA  LOC3   ; 取指定存贮单元数据
STA  SR       起动移存器
BRK
```

例　从存贮单元LOC3中取出数据，在T_2控制下移位输出。

```
LDA  #0
STA  ACR
LDA  #$14   ; $14="00010100"
STA  ACR    ; 置T2速率移位输出方式。
LDA  #$20
STA  T2L-L  ; 置T2
LDA  #0
STA  T2C-H  ; 启动T2
LDA  LOC3   ; 从指定存贮单元取数据。
STA  SR     ; 移位输出
BRK
```

上面两个程序都只输出了8bit 数据，如果要求继续输出许多单元的数据，程序应如何修改，请读者思考。

§4—3　6530　ROM—RAM—输入/输出及定时器(RRIOT)

6530芯片是美国ROCKWELL公司在1975年推出6502的同时，专为与其配合可以组成一个计算机而设计生产的。设计者在一个芯片内将1KByte(字节) 的只读存贮器(ROM)、64kByte(字节)的随机存贮器(RAM)、两个8bit输入输出口以及定时器组合在一起，就当时的技术来讲确实达到了很高的水平。但其中的ROM不能改写，只能在生产芯片时一次做好，这就很大程度上限制了它的使用。美国的KIM—1单板计算机就使用了二片6530，单板机的监控

程序就存放在6530的ROM中。以后ROCKWELL公司改进6530而推出6532(1977年)和6522(1979年)，后二者应用更为广泛，功能也更加完善。

图4—7是6530的内部结构框图。图4—8是6530的引腿分配图。显然有三条引腿即17～19腿是由六个信号共用的，对于引腿17 ($\overline{IRQ}$/PB7) 究竟是哪一个占用由编程确定；对于引腿18和19则

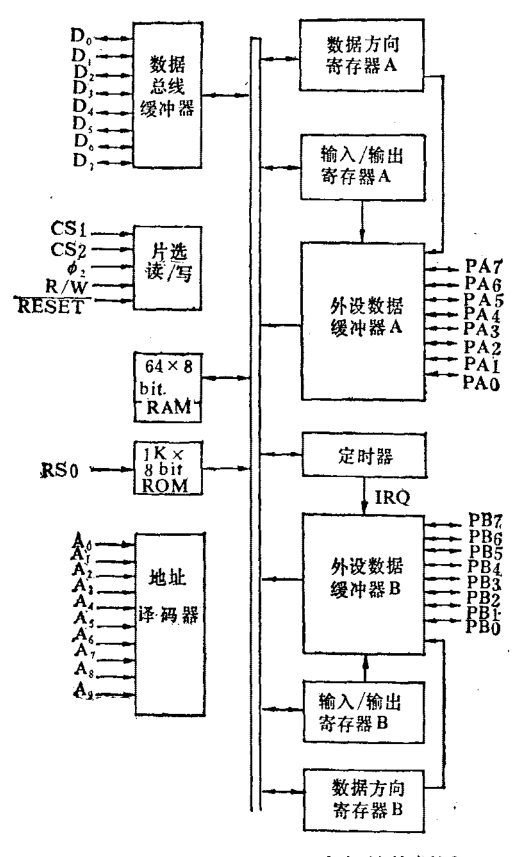


图4—7　6530内部结构框图

由工厂制造时来决定。

6530内部的定时器包括三个部分，即脉冲分频器、可编程 8bit 寄存器和中断控制逻辑。可编程寄存器可以被MPU读写，可写入

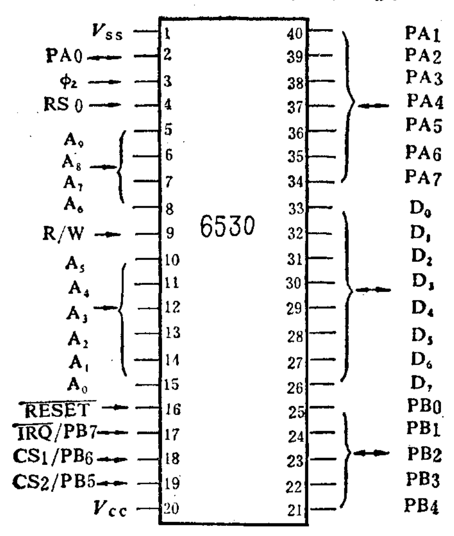


图4—8　6530引腿分配

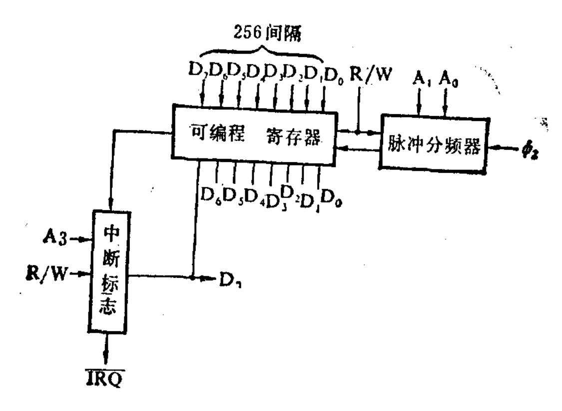


图4—9　6530内部定时器框图

256种不同的间隔，系统时钟ϕ_2经过脉冲分频器再进入可编程寄存器，它执行减量计数，当减到八位为全1时将使中断标志位置1，同时$\overline{IRQ}$变成0电平。脉冲分频器有四种分频系数可供选择即1、8、64和1024。$\overline{IRQ}$线是利用引腿17，如果PB7所对应的数据方向寄存器的位的内容为0，写操作时A3是1，则将使引腿17成为$\overline{IRQ}$信号的引线。脉冲分频器的系数则由A_0、A_1译码决定。

6530的两个输入输出8bit口，数据方向寄存器以及数据总线缓冲器都与6520很类似，这里就不再赘述。

6530有A_0～A_9十根地址线，除此之外还有一根ROM选择线RS0，以及附加的两根片选线（CS1、CS2）。片选线也可以是PB5和PB6，究竟是片选线还是输入输出口的引线是在工厂制造时选择的。片选两根线是相互独立的，即任何一根可指定为片选而另一根可指定为输入输出线。当然这些因素将使6530的使用很不方便。

究竟怎样安排6530中的ROM、RAM、输入输出口以及定时器的地址呢？这是比较麻烦的。要事先确定你设计的计算机将使用几片6530以及64k存贮空间如何分配，再与制造6530的工厂提出，然后他们也按你的要求制造6530芯片。现在介绍一个单片6530的地址分配情况。

图4—10是6530单片地址编码图，图中“*”处是工厂在制造时专门连接上的。从图可以看出ROM有A_0～A_9地址线以外还有一条ROM选择线，RAM除A_0～A_5之外还有一条RAM选择线，输入输出口四个寄存器除了A_0、A_1外还有入/出选择线；定时器A_0、A_1决定脉分频器系数（即00—1，01—8，10—64，11—1024）以及A3为1之外，还有一条定时器选择线。这些选择线分别同四个与门输出相连。由图可知：

ROM选择 = CS1·RS0

RAM 选择 = $\overline{CS1}$ · $\overline{RS0}$ · $\overline{A9}$ · A7 · A6

输入/输出选择 = $\overline{CS1} \cdot \overline{RS0} \cdot A9 \cdot A8 \cdot A7 \cdot A6 \cdot \overline{A2}$

定时器选择 = $\overline{CS1} \cdot \overline{RS0} \cdot A9 \cdot A8 \cdot A7 \cdot A6 \cdot A2$

由图4—10可以看出单片6530系统引腿19不是CS2仍然可以作为输入输出引腿PB5使用。

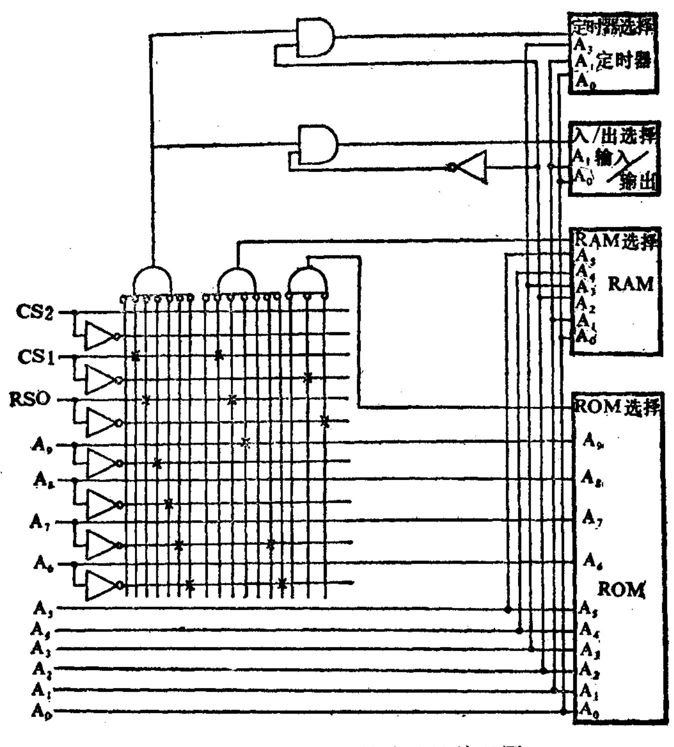


图4—10　6530单片地址编码图

§4—4　6532RAM—输入输出—定时器(RIOT)芯片

6532基本上就是一个没有ROM的6530，但它的随机存贮器

(RAM) 比6530大些，有128个字节。另外PA7线通过编程可以作为一个边沿检测输入线。在这种方式下一个有效的跳变边沿将使中断标志位置1,如果允许这将使引线$\overline{IRQ}$变成0电平，向MPU 提出中断请求。

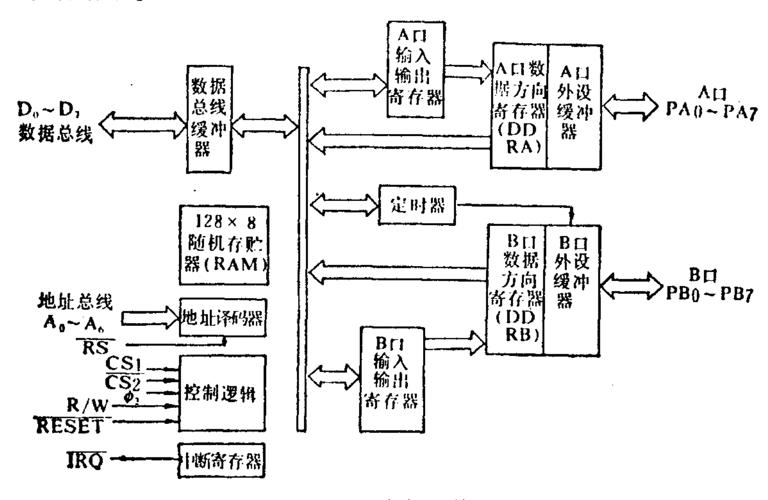


图4—11　6532内部结构框图

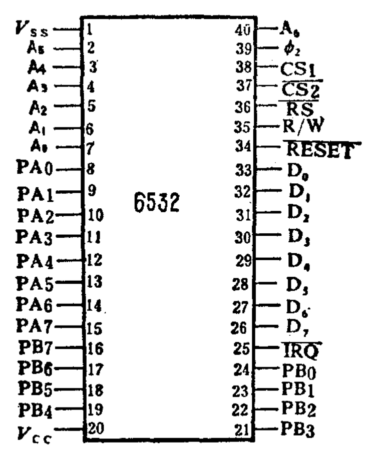


图4—12　6532引腿分配

由图4—11可以看出6532基本上是四个部分：即RAM、输入/输出口、定时器和中断逻辑。输入输出部分的功能与6520类似，这里就不必重复了。RAM共有128个字节，这是一个普通的静态存贮器，当CS1＝1，$\overline{CS2}=0$，$\overline{RS}=0$时可以通过地址A_0～A_6对每一个存贮单元寻址，而R/W（读/写）线则控制对它的读或写。R/W线为1电平即进行读操作，R/W线为0电平即为写操作。

6532的定时器部分同6530基本相同（见图4—9），依然是8bit可编程寄存器由MPU预先置数。分频器可对系统时钟ϕ_2选择1、8、64、1024倍分频。启动定时器以后可编程寄存器可进行减量计

表4—19　　　　6532寻　址

$\overline{CS2}$	CS1	$\overline{RS}$	A4	A3	A2	A1	A0	R/W	功　　能
0	1	0	–	–	–	–	–	–	再加上地址线A6、A5对RAM寻址
0	1	1	–	–	0	0	0	–	A口输入输出寄存器
0	1	1	–	–	0	0	1	–	A口数据方向寄存器
0	1	1	–	–	0	1	0	–	B口输入输出寄存器
0	1	1	–	–	0	1	1	–	B口数据方向寄存器
0	1	1	1	*	1	0	0	0	写定时器。分频系数为1
0	1	1	1	*	1	0	1	0	写定时器。分频系数为8
0	1	1	1	*	1	1	0	0	写定时器。分频系数为64
0	1	1	1	*	1	1	1	0	写定时器。分频系数为1024
0	1	1	–	*	1	–	0	1	读定时器
0	1	1	–	–	1	–	1	1	读中断标志
0	1	1	0	–	1	**	***	0	写检出边沿极性控制

注："—"表示可以为0或1

数，当减到零时将中断标志置1。如果 $A_3=1$则允许中断，这时 $\overline{IRQ}$线将变成0；反之$A_3=0$中断将被禁止。分频系数由A_0、A_1控制。如果A_1、A_0的值分别是00、01、10、11则分频系数对应为1、8、64、1024。表4—19列出了6532的寻址。

显然，所有的选择中如果CS＝1，$\overline{CS2}=0$，$\overline{RS}=0$是选择了RAM，而$\overline{RS}=1$即选择到其他寄存器；所以$\overline{RS}$线叫RAM选择线。$A_2=0$即选到输入输出寄存器；$A_2=1$即是选到定时器或中断逻辑部分。定时器的分频系数由A_1、A_0决定。表中“*”号表示：当“*”＝1允许定时器中断，即当定时器中断标志位被置1时，将使引线$\overline{IRQ}$变0电平，向MPU提出中断请求；反之“*”＝0则禁止定时器中断，即当定时器中断标志位被置1时，引线$\overline{IRQ}$依然保持1电平，不向MPU提出中断请求。表“* *”号表示控制PA7中断输入线；如果* *＝1则PA7为中断输入线；如果* *＝0则禁止PA7中断。“* * *”号则是确定检测的边沿是正跳还是负跳；如果* * *＝1——正跳，* * *＝0——负跳。

前面已经谈过，6532可以有两个中断源：一是定时器，另一个则是由引线PA7输入的信号。它们各自对应着一个中断标志位。当我们按表4—19所列出的地址读中断标志时，定时器的中断标志位将送到数据总线的 Bit7，而 PA7 的中断标志则送到数据线的 Bit6。

如前所述，6532的控制作用同6522有些地方不同，这主要是由于6532不像6522那样专门设有中断标志、中断允许、外设控制和辅助控制等寄存器，它的控制作用不是去对这些寄存器写入控制数据，而是通过改变地址来实现，也就是说，控制信息表现在某些地址位的电平上，这是同6522完全不同的。以这种方式进行控制时，数据总线上的数据为何值反到是与控制作用无关的。

§4—5 异步通信接口适配器(ACIA)6850

美国MOTOROLA公司出品的6850同样可以与6502相配合，实现异步串行通信。它的主要功能是完成把MPU6502送来的八位并行数据变为串行数据输出；反过来它也可以把串行输入的数据转换成八位并行数据送往6502。

图4—13和4—14分别画出了6850的逻辑框图及管腿图。从图4—13可以看出：6850有八条数据总线以及数据总线缓冲器，以便同MPU6502交换数据，为此它还有片选与读写控制部分。这些同前面几节谈到的那些芯片是很相似的。6850还包括发送与接收串行数据的部分，这主要是发送与接收移位寄存器及其控制电路。发送数据线向外部发送串行数据，接收数据线将外部串行数据送进6850。数据移位的速率取决于外部接入的发送和接收时钟，发送和接收控制器是保证发和收都可以按控制寄存器来完成各种功能，而控制寄存器则可接受MPU6502写入的控制数据——我们常称为控制字。6850还有状态寄存器，它的八个位分别反映着6850本身以及同它相联系的调制解调器的工作情况，状态寄存器只能由6502读出。

由图4—13可以看出6850有发送数据寄存器、控制寄存器、接收数据寄存器以及状态寄存器四个寄存器，前二者只能由6502向它们写入数据，而后者只能由6502读出它们的数据。

图4—13的左上角是6850的片选与读写控制逻辑部分，由图可以看出它包括三根片选线（CS_0、CS_1、$\overline{CS_2}$），它们一般与地址线相连，使6850在系统中使用时可以很方便地对它进行寻址。读/写控制线一般与MPU6502的R/W输出线相连，使6502可对它进行读写操作。RS是寄存器选择线。前面已提到6850虽包括四个寄存

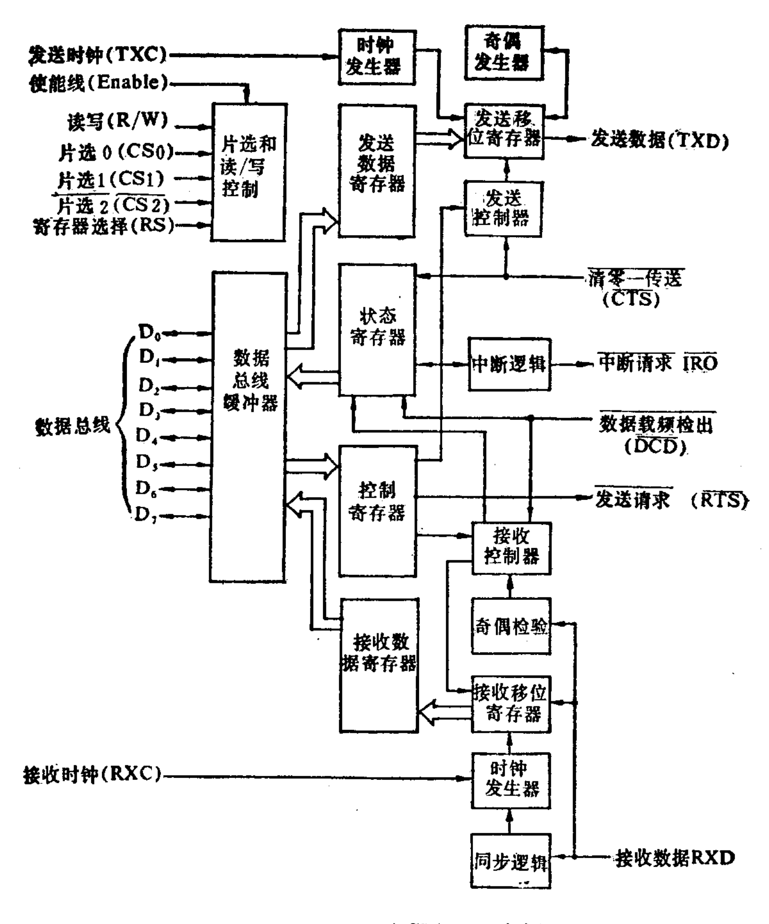


图4—13 ACIA6850方框图

器，但能读或能写的各为二个，因此可以用RS线和读写控制线相配合完成对四个寄存器的寻址，如表 4—20 所示。使能线(Enable)一般同系统时钟ϕ_2相连，6850仅在ϕ_2为1时才与 MPU6502交换数据。

图4—13中右边除了我们前面已提到的发送数据和接收 数 据

表4—20

R/W	RS	寄存器名称
0	0	控制寄存器
0	1	发送数据寄存器
1	0	状态寄存器
1	1	接收数据寄存器

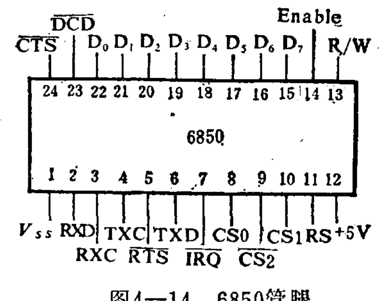


图4—14 6850管腿

线以及中断请求（$\overline{IRQ}$）线以外，尚有几根6850与调制解调器相联络的线。对于它们我们分别说明如下：

请求发送（$\overline{RTS}$）：这是一个输出信号，低电平有效。MPU可以通过数据总线对6850的控制寄存器的Bit5 和Bit6写入数据来控制这根输出线。6850通过它与调制器相联系，输出低电平即通知调制器MPU已准备好请求发送。

清零—传送（$\overline{CTS}$）：这是一个输入信号，低电平有效。它是由调制器来的信号，是调制器作为对$\overline{RTS}$信号的响应。当它为低电平时6850发送信号。

数据载频检出（$\overline{DCD}$）：这是一个输入信号，低电平有效。它的作用很类似前面的$\overline{CTS}$，不过它是由解调器来的信号。如果$\overline{DCD}$信号由低变高则禁止6850的接收部分工作。只有它为低时，6850接收部分才接收数据。

应当指出，发送数据是用发送时钟（TXC)的负沿进行同步；而接收数据则是用接收时钟（RXC）的正沿进行采样。

以下我们对6850的控制寄存器和状态寄存器进行说明：

表4—21列出控制寄存器(CR）所有各位的分配。其中Bit7是6850的接收部分的中断控制位，它为1时表示允许中断，为0即禁止中断。所谓允许中断即当状态寄存器（见表4—23）的Bit0—接收数据寄存器满——为1，或者数据载频检出($\overline{DCD}$）信号由低变

高时，6850的$\overline{IRQ}$（中断请求）线将由高变低，因此可向MPU6502提出中断申请。Bit6和Bit5是对6850的输出线$\overline{RTS}$（请求发送）

表4—21　　控制寄存器

位	7	6	5	4	3	2	1	0
功能	接收中断允许	发送控制2	发送控制1	字选择3	字选择2	字选择1	计数器分频选择2	计数器分频选择1

的控制，同时也是对状态寄存器（见表4—23）的Bit1——发送数据寄存器空——的状态和6850的另一根输出线$\overline{IRQ}$（中断请求）之间关系的控制。我们用表4—22列出了四种不同的情况所对应的控制作用。所谓发送允许中断，即当状态寄存器的Bit1为1时，则将使6850的$\overline{IRQ}$引线为低电平，使其向MPU6502申请中断。而发送禁止中断之意即当状态寄存器的Bit1变为1时，仍然不能使$\overline{IRQ}$线变低。控制寄存器的Bit4、Bit3、Bit2三位码构成八种不同的情况，分别控制着发送数据的格式。6850是起止式异步串行通信芯片，它的基本格式如图4—15。由图可以看出它有一个起始位，然后接下去是数据——当然这就是你所要传送的信息。在数据传输

起始位	数据	奇偶位	1位或2位停止位

图4—15　6850通信格式

的过程中往往由于干扰而使数据传输出错，为了提高通信的可靠性，我们在有用的数据后面再加上一位码，这位码我们称它为奇偶位。加上它以后使传输的码中1的个数成偶数或成奇数，具体是偶数还是奇数要由设计通信系统的人来规定。作了这样的规定以后，如果有某位码在传输过程中由于干扰使其由1（或0）变成0（或1），这就破坏了码位的奇偶性，在接收端我们可以通过奇偶检验而发觉信息在传输中产生了错误。格式中的停止位表示这一个字

表4—22　　控制寄存器（CR）某些位的详情

CR6		CR5	功　　能
0		0	$\overline{\text{RTS}}$线输出为低电平，发送禁止中断
0		1	$\overline{\text{RTS}}$线输出为低电平，发送允许中断
1		0	$\overline{\text{RTS}}$线输出为高电平，发送禁止中断
1		1	$\overline{\text{RTS}}$线输出为低电平。在发送数据输出端发送间隔电平，发送禁止中断
CR4	**CR3**	**CR2**	**功　　能**
0	0	0	7位＋偶数奇偶位＋2终止位
0	0	1	7位＋奇数奇偶位＋2终止位
0	1	0	7位＋偶数奇偶位＋1终止位
0	1	1	7位＋奇数奇偶位＋1终止位
1	0	0	8位＋2终止位
1	0	1	8位＋1终止位
1	1	0	8位＋偶数奇偶位＋1终止位
1	1	1	8位＋奇数奇偶位＋1终止位
CR1		**CR0**	**功　　能**
0		0	1/1
0		1	1/16
1		0	1/64
1		1	总清零

符传送已经结束。图4—15的格式中究竟每一位占多少时间则由发送时钟的周期以及CR0、CR1决定。表4—22可以看到也可以不加奇偶位，这些是可以通过 MPU6502 对控制寄存器的写入来控制的。表4—22中的CR0和CR1决定着对发送脉冲的分频次数。例如当用发送时钟为1760赫，选择分频为1/16时，则数据发送的速率为

$$1760/16 = 110位/秒$$

通信中把传送速率的单位叫做波特（BAUD)。每秒钟传一位二进制码就叫一波特，所以上面的例子即为110波特。

最后我们强调指出，6850没有复位线，为保证它可靠工作，在使用它之前MPU6502应将CR0、CR1均写入1，以实现对6850的总清零。这一点在编程时一定要记住。

以下我们再来谈谈6850的状态寄存器。如前所述，这个寄存器反映了6850接收和发送部分的工作情况，而且它只允许MPU读出它的数据，而不可能对它进行写入数据。它的Bit0是接收数据寄存器满（RDRF）位。当它为1时，即表示接收数据寄存器已经收满了数据，当MPU6502对接收数据寄存器进行读操作之后，该位将为0。状态寄存器的Bit1是发送数据寄存器空（TDRE)位，它为1表示发送数据寄存器已无数据可发送了，应写入下一个数据，它为0即表示发送数据寄存器中的数据尚未被传送完。$\overline{DCD}$位即数据载频检出位，它是状态寄存器的Bit2，它反映了6850的一根输入线（$\overline{DCD}$线）的状态。如前所述这根线是由解调器送来的，当没有载频信号输入时，解调器来的信号应为高电平，则该位应为1。这样如果控制寄存器的Bit7为1，则6850的 $\overline{IRQ}$ 线将输出0电平，向MPU6502申请中断。$\overline{DCD}$位可由读出状态寄存器清零，或引线$\overline{DCD}$ 变低电平也将使它清零。状态寄存器的 Bit3是清零——发送位。它很类似Bit2，不过它反映的是引线 $\overline{CTS}$的状态。这根引线如前所述是接受调制器送来的“允许发送”的信号。这根

线为高电平，Bit3（$\overline{\text{CTS}}$位）即为1，这根线为低电平，$\overline{\text{CTS}}$位即为0。如果该位为1将禁止TDRE位工作。表4—23列出状态寄存器的Bit4是帧结构错(FE)位，如果接收到的信号起始位或停止位有错，该位即为1。接收越界(OVRN)位即Bit5，它表示MPU6502尚未将6850已收到在接收数据寄存器的数据读出，接收器又收到了下一个数据（显然将使原来的数据全部或部分丢失）。如果发生了这种情况OVRN位将为1。奇偶位错误(PE)位表示接收到的数据奇偶性有错误。状态寄存器的Bit7是中断请求(IRQ)位，它反映6850的

表4—23　　　　　6850状态寄存器（SR）

位	7	6	5	4	3	2	1	0
功能	(IRQ) 中断请求	(PE) 奇偶性错误	(OVRN) 接收越界	(FE) 帧结构错	($\overline{\text{CTS}}$) 清零—发送	($\overline{\text{DCD}}$) 数据载频检出	(TDRE) 发送数据寄存器空	(RDRF) 接收数据寄存器满

$\overline{\text{IRQ}}$引线的状态。当$\overline{\text{IRQ}}$引线为低电平时，该位即为0；而$\overline{\text{IRQ}}$引线为高电平时，该位将为1。

为说明6850的工作，下面我们列举两个简单的例子。

例1　通过6850接收一个串行数据，并将它放在LOC1单元中。

```
       LDA   #$03    ; $03=00000011
       STA   ACIACR  ; 将控制寄存器Bit1、Bit0写入11，使
                       6850总清零
       LDA   #$45    ; $45=01000101
       STA   ACIACR  ; 详细在下面说明
WAITD  LDA   ACIASR  ; 读6850状态寄存器
       LSR   A       ; 已收满数据吗?
       BCC   WAITD   ; 未满，再等待
       LDA   ACIADR  ; 已满，则读数据接收寄存器
```

```
        STA   LOC1        ; 存到LOC1单元
        BRK
```

上面程序中第四行将数据01000101写入6850控制寄存器，由表4—21和4—22我们知道：

Bit7 = 0　禁止接收中断

Bit6 = 1

Bit5 = 0　使 $\overline{\text{RTS}}$ 引线为高电平并禁止中断

Bit4 = 0

Bit3 = 0

Bit2 = 1　即7位数据＋奇数奇偶性＋2位停止位

Bit1 = 0，Bit0 = 1即分频系数为16。

如果接收的是传输速率为110波特的电传机来的数码，接收时钟必须是16×110 = 1760赫。

例2　将LOC2存贮单元的数据，通过6850串行发送。

```
        LDA   #$03
        STA   ACIACR      ; 将6850总清零
        LDA   #$45
        STA   ACIACR      ; 见上例
        LDA   #$02
WAITR   BIT   ACIASR      ; TDRE位为1吗?
        BEQ   WAITR       ; 不为1，等待
        LDA   LOC2        ; 若为1（即发送数据寄存器空),即
                          ; 从LOC2单元取出数据
        STA   ACIADR      ; 将数据写入发送数据寄存器
        BRK
```

同例1一样，如果数据的发送给电传机，发送时钟应为1760赫。



第五章　应用举例

前面几章我们已就6502微处理机的指令系统、汇编语言程序设计、接口芯片和典型的微计算机进行了较系统地介绍。本章拟挑选一些具有代表意义的应用实例进行分析，举出这些例子的目的不仅是要告诉你在具体的例子中怎么应用，而且还打算要启发你自己去解决一些实际问题。

本章开始介绍键盘与打印机的接口，在这里介绍的是最简单的键盘与打印机，但正是由于它们简单才便于使初学者更易于理解。其实，对简单的输入输出的理解将会给进一步学习带来好处。接下去介绍的是摩尔斯电码和频选电话发生器。这二者虽可以直接用于实际应用，但我们在这里选择它们是着重于6522的应用以及用软件方法完成码转换这个带有一般性问题的技巧。第三节是APPLE Ⅱ机的绘图与发声。这可以使你在使用APPLE 机设计游戏和辅助教学或其它应用程序中得到启发和帮助。往下是AIM-65电子钟程序。它是一个包含有中断服务程序、子程序、主程序的一个完整的程序，这里你可以进一步体会到中断这种输入输出方法的好处。最后我们介绍了D/A和A/D变换。它们是许多计算机实际应用都会遇到的问题，我们由浅入深逐步地予以说明，对读者将会有所帮助。

§5—1　键盘与打印机

键盘与打印机是最常用的输入输出设备，这里介绍它们如何与计算机联接。虽然我们为了方便键盘的按键数目是十六个，打

印机也是以一种微型打印机作例子，但对于其它键盘或打印机其基本原理还是大体上相同的，因此这里举的例子是有实际意义的。

首先我们介绍采用矩阵线路的键盘，如图5—1所示。图中采用6522的A口作输出口，低四位的输出线画成横排线，高四位画成纵列线，构成了一个矩阵形式的电路。在横排线和纵列线的交叉处连接十六个按键。每个按键正好定义为一个十六进制数字(如图5—1右边方格)。

我们使6522输出二进制数**00001111**，如果十六个键当中没有一个键被按下，则当我们读入A口的数码时，依然是**00001111**，但如果有某一个键被压下，则从A口读回来的数值就不再会是**00001111**了。例如B键被压下，由于纵列线3与横排线2相连通，这将使横排线2也为**0**电平。因此A口读回的数值将会变成**00001011**。低四位中的bit 2变成了**0**，可以断定压下去的键是在横排线2上。如果我们这时使6522的A口输出**1111011**，再从A口读回数值。由于B键按下，横排线2仍然同纵列线3相连，故读回的数值变成了**01111011**。高四位bit 7变成了**0**，可以断定压下的开关一定在纵列线3上。用这种办法可以确定任何一个被压下的键的位置。从上面的例子我们可以看出，最后读回A口的数值是**01111011**（$7B)，这当中两个**0**，高四位中的一个表示纵列线位置，低四位中的一个则表示横排线的位置。依此我们可以确定：如果**0**键被压下，最后从A口读回的数值一定是**11011110**($DE)，键E压下是**01111110**（$7E)。由此我们可以列出表5—1，表中即表示了我们按图5—1连接和定义的十六个键所对应的识别编码及ASCII码（都是用十六进制数表示)。

以下我们为图5—1这个键盘设计一个子程序，这个程序可以将压下的键转换成其所对应的ASCII码。

```
        LDA   #$FF
        STA   DDRA  ；置A口所有的线为输出线
```

表5—1　　键盘码表

字　符	识别码	ASCII码
0	DE	3 0
1	ED	3 1
2	DD	3 2
3	BD	3 3
4	EB	3 4
5	DB	3 5
6	BB	3 6
7	E 7	3 7
8	D 7	3 8
9	B 7	3 9
A	7 7	4 1
B	7 B	4 2
C	EE	4 3
D	BE	4 4
E	7 E	4 5
F	7 D	4 6

0	1	2	3	
C	O	D	E	0
1	2	3	F	1
4	5	6	B	2
7	8	9	A	3

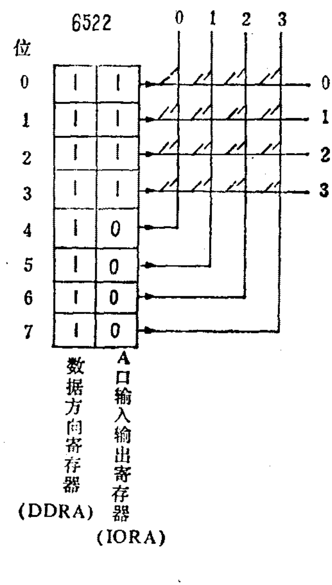


图5—1　矩阵式键盘连接图

```
LDX   #$0F
STX   IORA   ；向A口输出00001111
LDA   IORA   ；读回A口，确定横排线位置
ORA   #$F0   ；高四位成全1
STA   IORA
```

```
        LDA  IORA        ；再次读回A口，确定纵列线位置
LOOP    CMP  TAB, X      ；与键F的识别码比较
        BEQ  CNVT        ；如果相等，即跳转
        DEX
        BPL  LOOP        ；如果X≥0则继续查表比较
        BMI  NOFND       ；如果X<0则表示没有找到
CNVT    LDA  ASCT, X     ；A累加器中即为所按下键的
                          ASCII 码
        RTS
NOFND ...                ；没有找到应转出错指示
TAB BYTE  $E7, $D7, $B7, $77, $EB, $DB,
          $BB, $7B, $ED, $DD, $BD, $7D,
          $EE, $DE, $BE, $7E
ASCT BYTE $37, $38, $39, $41, $34, $35, $36
          $42, $31, $32, $33, $46, $43, $30
          $44, $45
```

图5—2是程序框图。程序中标号TAB以下十六个字节存放着识别码；标号 ASCT 以下十六个字节则对应存放着字符 0～F 的ASCII码。两个表格中的数值都是十六进制数表示的。如果操作键盘的人同时按下二个或二个以上的键，则在表中找不到所对应的码，应转到出错指示子程序。另外应指出的是，本程序中没有加入防止按键接触不良的反弹跳延迟程序。

对于小键盘仍可采用另外的方法联接，如图5—3。图中采用四线——十六线译码器（74154）将 6522 的B口输出线PB0～PB3进行译码。74154 是以反码形式输出，当 PB3～PB0＝0000 时，74154的端子1输出为0电平，而其它输出端子均为1；当PB3～PB0＝0001，则端子 2 输出为0电平，其它输出端子为1电平，同样当PB3～PB0＝1111则端子17输出为0电平，其它输出端为1。由图5

—3可见键0～键F一共十六个键，它们的一头分别与74154的十六

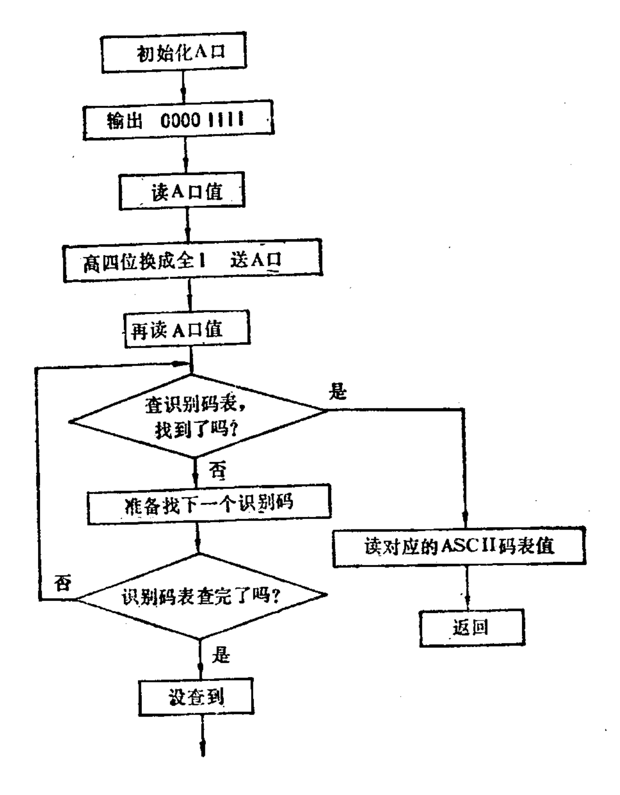


图5—2 程序框图

个输出端顺序相连；另一端则都接在一起，再通过一电阻连到+5V电源，同时连到6522A口的PA7端。A口定义为输入口。图5—3再配合一段子程序，可以完成转换成ASCII码的功能。

显然，如果所有的键不压下时，PA7应为1电平。当键0压下时，只有当PB3～PB0输出0000时，PA7为0，而输出别的值时，PA7仍然为1；而当F键压下时，仅当PB3～PB0输出1111时，PA7为0。因此可以通过PB3～PB0输出，同时测定PA7的输入来判定

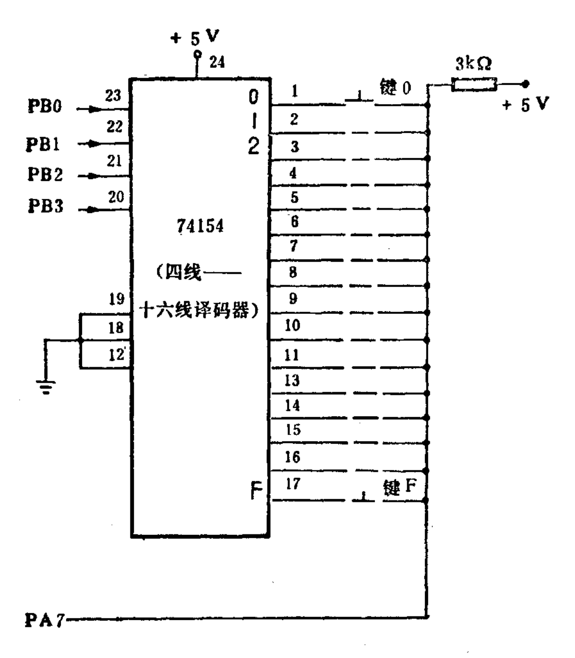


图5—3 小键盘的另一种连接法

究竟是哪个键被压下。图 5—4 是将所压下的键转换成相应的ASCII码的程序框图。这个图中有几处需要着重说明：首先是一开始就判键压下了吗？这是为了防止上一次压键过久尚未抬起，就已进入了本子程序，从而避免了一次压键多次转换的可能性。其次图中加入了反开关弹跳的程序。使用一延时程序，延迟时间可设定为几十毫秒，加上这部分以后，使得只有压稳了的键才能转换。从而保证了转换的可靠性。

程序如下：

```
LDA   #0
STA   DDRA      ；置6522A口所有的线为输入线
LDA   #$FF
```

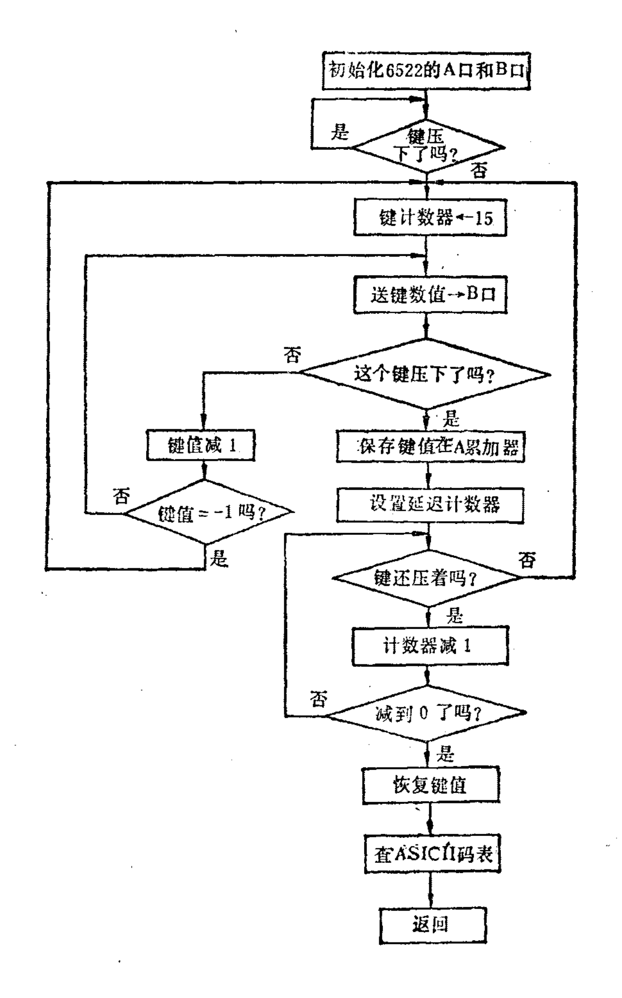


图5—4 程序框图

```
        STA DDRB    ；置6522B口所有的线为输出线
START   BIT  IORA   ；读A口，判PA7为0码？
        BPL START   ；PA7＝0,表示上一次键仍然继续压
                      下，应等待抬起
```

```
RSTART  LDX #15       ; 置键计数器
NXTKEY  STX IORB      ; 输出到B口
        BIT  IORA     ; 测PA7是为0码?
        BPL BOUNCE; 是0，则跳转到开关反弹跳程序。
        DEX           ;准备查下一个键
        BPL NXTKEY; 键计数器值不为负值则再查下一
                      个键
        BMI RSTART; 键计数器值为负， 再重新开始。
BOUNCE  TXA           ; 送键值到A口保存
        LDY #$12      ; 若延迟50ms所需设置的值
LP1     LDX #$FF
LP2     BIT IORA      ; 键还压着吗?
        BMI RSTART  ;如果没压着就不必转换了。跳转到
                      重新开始。
        DEX
        BNE LP2
        DEY
        BNE LP1       ; 延时50ms
        TAX
        LDA TAB, X  ; 将ASCII码表中所对应的值送累
                      加器A
        RTS
    TAB BYTE  $30, $31, $32, $33, $34, $35,
              $36, $37, $38, $39, $41, $42,
              $43, $44, $45, $46
```

如果没有键压下，程序将一直等待下去；如果压下几个键，程序将只取其中的某一个进行转换，并不指示出错。

下面我们谈谈打印机的连接问题，这里仅以一种微型打印机

为例子。假定打印宽度是每行20个字符，一般打印机本身都带有一块小电路板，这就是它的接口，这个接口将对打印头、走纸以及打印机的其他机械部分进行管理。因此这种微型打印机是可以同任何一个计算机的并行口（PIO）相连接的。本例题就是采用6502单板计算机（如AIM-65或SYM-1）的6532和6522同它相联。如果使用的打印机不同，或单板机不同将会有不同的程序，但是程序的逻辑应当是基本上相同的。

这种微型打印机一般只可打印六十四种不同的字符，因此它的字符用六位二进制数表示（即ASCII码中的字母、数字以及某些常用符号）。打印机接口将有几条信号线，即“起动打印”、“字符请求”和“打印机忙”。以下我们写一个程序向打印机送20个字符，然后回车换行。首先由计算机发出起动打印机信号，然后顺序送20个字符。每送出一个字符前，计算机应等待打印机接口的“字符请求”信号，作为这个信号的响应，计算机送出一个字符，

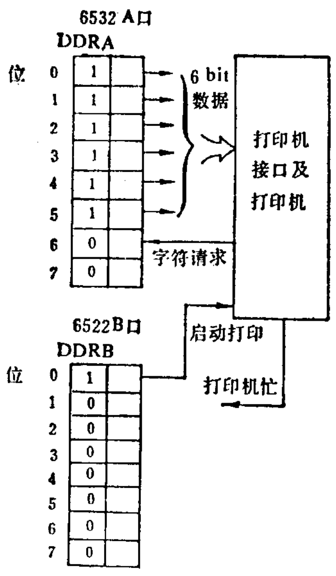


图5—5 微打印机的连接

即在六根数据线上提供一个字符的ASCII码。然后等待打印机取走这个数据，再送下一个字符。待20个字符全部送完以后，再送一个空格符，打印机将完成回车换行。

图5—5是计算机到打印机接口的连接图。我们用6522B口的PB0向打印机提供“起动打印”信号，用6532A口PA0～PA5提供6 bit数据，PA7作为输入，接收打印机的“字符请求”信号，不用“打印机忙”线，而用一段延时程序代替。

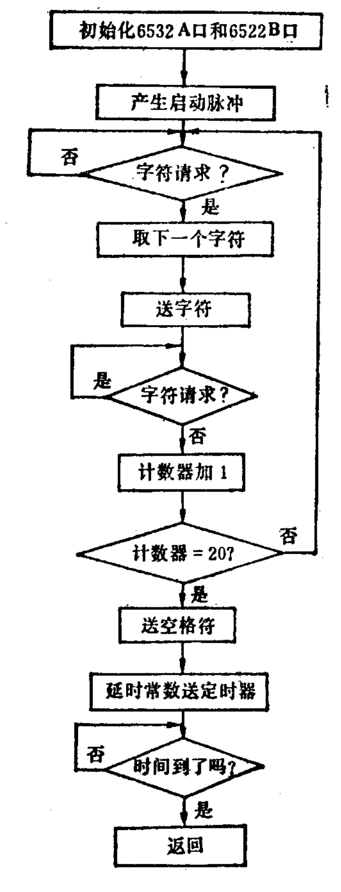


图5—6　打印程序框图

```
LINE LDA  #$3F
     STA  DDRA   ；初始化6532A口
     LDY  #1
     STY  DDRB   ；初始化6522B口
     STY  IORB   ；送起动信号
```

```
      DEY
      STY IORB    ；起动信号回0电平
TST1  BIT  IORA   ；读A口PB6"字符请求"信号
      BVS TST1    ；"字符请求"吗？是，应为0
      LDA (00),Y；取字符。详见图5-7
      STA IORA    ；送打印机
TST2  BIT  IORA   ；查"字符请求"信号
      BVC TST2    ；如果没有，继续等待
      INY         ；准备取下一个字符
      CPY #$14    ；20个到了吗？
      BNE TST1    ；不到，应再重复上面过程
      LDA #$20    ；取空格符ASCII码
      STA IORA
      LDA #$30
      STA T1024   ；置6532定时器分频系数为1024
TTIM BIT  TIMFLG；查定时器中断标记
      BPL  TTIM   ；标记为0再等待
      RTS
BUFF BYTE $30, $31, $32, $33, $34, $35
      BYTE $36, $37, $38, $39, $41, $42,$43,
           $44, $45, $46, $47, $48, $49,$4A
```

程序中的字符表存在某个缓冲区，而将表的首地址存放在零页的00和01两个存贮单元。低字节放00单元，高字节放01单元。指令LDA(00)，Y将按图5—7寻址。

程序中T1024，，应查看6532的寻址表（即表4—19）。要求A0＝A1＝1,A4＝A3＝A2＝1。6532定时器中断标记地址A0＝A2＝1，A3、A1、A4可为0或1。具体地址值应由计算机中MPU的地址线同6532如何连接决定。程序中的其它符号如IORA、IORB、

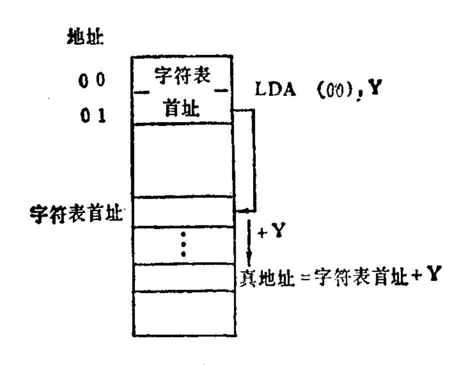


图5—7 间址变址存取

DDRA、DDRB的具体地址值应由具体的机器来定。

§5—2 摩尔斯电码和频选电话发生器

电报通信中广泛使用摩尔斯电码，它是由电报员操纵按键控制长短（划、点）节奏来代表每个字符（英文字母、数字及各种打印符号）信息。这一套编码的规定如表5—2所示。由表可以看出：英文打字机键盘上的每一个字符几乎都有对应电码；电码的基本元素是·(点)和—(划)。我们如果将电码产生的电信号用一个耳机（或喇叭）接收的话，将可以听到"滴""嗒""滴""嗒"的声音与一定的节奏，所谓节奏就是点划不同组合而形成音响。显然操纵电键自如地将打印机的每个字符转成电码的电报员是需要经过长期专门训练的。这里我们列举一个例子，使用单板计算机（如SYM-1）可将存贮器中存放好的字符（ASCII码）转换成摩尔斯电码，如果接上一付耳机（或喇叭）还可以听到"滴嗒"声。

首先我们谈谈将摩尔斯电码换成二进制码的规则。前者的基

表5—2　　　　　　**摩尔斯电码转换表**

字符	摩尔斯电码	ASCII码	转换编码	字符	摩尔斯电码	ASCII码	转换编码
,	- - • • - -	2C	73	G	- - •	47	0E
—	- • • • -	2D	31	H	• • • •	48	10
•	• - • - • -	2E	55	I	• •	49	04
/	- • • - •	2F	32	J	• - - -	4A	17
0	- - - - -	30	3F	K	- • -	4B	0D
1	• - - - -	31	2F	L	• - • •	4C	14
2	• • - - -	32	27	M	- -	4D	07
3	• • - -	33	23	N	- •	4E	06
4	• • • • -	34	21	O	- - -	4F	0F
5	• • • • •	35	20	P	• - - •	50	16
6	- • • • •	36	30	Q	- - • -	51	1D
7	- - • • •	37	38	R	• - •	52	0A
8	- - - • •	38	3C	S	• • •	53	08
9	- - - - •	39	3E	T	-	54	03
?	• • - - • •	3F	4C	U	• • -	55	09
A	• -	41	05	V	• • • -	56	11
B	- • • •	42	18	W	• - -	57	0B
C	- • - •	43	1A	X	- • • -	58	19
D	- • •	44	0C	Y	- • - -	59	1B
E	•	45	02	Z	- - • •	5A	1C
F	• • - •	46	12				

本元素是“划”、“点”，后者的基本元素是1、0。现在我们将二者对应起来，即“划”（—)对应于1，“点”(•)对应于0。例如字母A的电码是“•—”，就对应二进制数码01；B是“—…”,则对应二进制数码是1000。为了识别方便我们在每个二进制数码的左边都加上一位1作为起动代码。另外，我们还规定不足八位二进制数的左边全部加上0，凑齐八位二进制数。这样字符A的摩尔斯电码“•—”即换成了二进制数码00000101($05)，B即成了00011000($18)。按照这个规则我们列出表5—2最右边的一列，表中全部用十六进制数表示，这一列数码我们称为摩尔电码的转换码。

一般我们采用送往耳机一个方波即可使耳机发声，方波的周期T 的倒数$\frac{1}{T}$即声音的基频。我们可以采用6522（详见§4—2）的定时器1工作在连续运行方式下，即可产生一个方波。方波的半周期可通过预先写入定时器1的数值所决定。至于方波产生的延续时间可以通过对定时器的控制来实现。具体地说我们可以让6522定时器1一会儿起动，一会儿停止，起动状态和停止状态的延续时间则通过一个延迟程序来控制。我们假定"点"的发声延续时间为一个单位时间，则"划"就是三个单位时间，字符之间的停顿间隔为三个单位时间，"点""划"之间的间隙为一个单位时间，空格符为七个单位时间。至于一个单位时间具体是多少秒，由使用者自行决定。我们的延迟子程序是：

```
DELAY LDA SPEED
   D2 LDX #$FA
   D1 DEX
      BNE D1
      SEC
      SBC #$01
      BNE D2
      DEY
      BNE DELAY
      RTS
```

程序框图如图5—8所示。如果送入寄存器X的值取$FA，则这个延迟子程序的总延迟时间是：

(Y寄存器的值)×(SPEED单元的值)×0.001秒。

因此我们可以将Y的值取1——"点"及间隙发声时间，Y=3——"划"发声时间，Y=7则是字符间隔时间，而SPEED单元的取值则决定一个单位时间的长短。例如(SPEED)=250，则一个

单位时间为1/4秒。

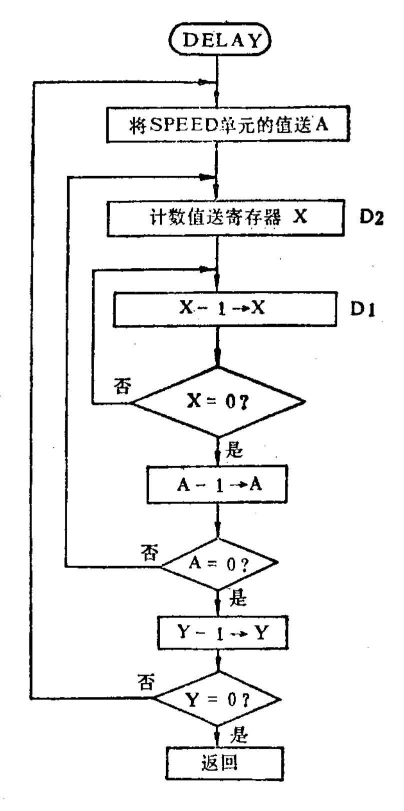


图5—8 延时子程序框图

至于对6522的控制可用下面这一段程序来完成：

```
SEND  LDA  #$C0    ; $C0=11000000，使定时器1为
                     连续运行方式，且6522 B口的
                     PB7引线上输出方波
      STA  ACR
      LDA  #$0
      STA  T1L-L   ; 将00000000写入定时器1低八位
```

```
                           锁存器
        LDA   #$04       ；将00000100写入定时器1高八位
        STA   TJL-H        锁存器，因此方波的频率约
                           500Hz
        STA   TIC-H      ；起动定时器1，PB7引腿开始输
                           出方波。
```

如果要定时器1方波停止输出应：

```
        LDA   #0
        STA   ACR
```

由于这将使6522的辅助寄存器的Bit7、Bit6全为0，这将禁止定时器1的连续运行工作，因此方波停止输出。

如果在输出方波的同时我们要使一个电键闭合，则应在启动定时器1之后加上：

```
        LDA   #$01
        STA   IORB
```

6522 B口的Bit0通过晶体管将使一个直流继电器吸合，方波停止，电键断开则应在停止方波之后加上：

```
        STA   IORB
```

B口的Bit0输出为0电平，继电器当即释放。

摩尔电码发生器子程序如下。在执行这个程序之前，主程序已将需要发出的电文的某个字符的ASCII码放在累加器A中。这个子程序的任务是将A中这个字符的摩尔斯转换码（见表5—2）找到，再按它发出摩尔斯电码。

```
MORSE   CMP   #$20      ；如果A中放的是空格符则转到
                          空格子程序
        BEQ   SPACE
        CMP   #$2C      ；如果A中的码小于 $2C、则
                          返回
```

```
        BCC   EXIT
        CMP   #$5B
        BCS   EXIT       ；如果A中的码大于$5A则返回
        TAX              ；将A中的值送到变址寄存器X
        LDA   TABLE-$2C,X ；取出摩尔斯转换码送A
        LDY   #$08       ；置计数值
        STY   COUNT
STARTB  ASL   A
        DEC   COUNT
        BCC   STARTB     ；将A中的转换码左移，一直到
                           起动位
        STA   CHAR
NEXT    LDA   CHAR
        ASL   A          ；现在移出的是摩尔斯电码(1=
                           "划"，0="点")
        STA   CHAR
        LDY   #$01
        BCC   SEND       ；如果C=0，则按一个单位时间
                           的延时发声
        LDY   #$03       ；否则按三个单位时间发声
SEND    LDA   #$C0       ；6522定时器T1为连续运行工
                           作方式；PB7输出方波
        STA   ACR
        LDA   #$FF
        STA   DDRB       ；使6522 B口所有线为输出线
        LDA   #$0
        STA   TIL-L
        LDA   #$04
```

```
        STA   TIL-H     ; 6522 PB7 输出的方波频率 约
                          500Hz
        STA   TIC-H     ; 启动定时器T1
        LDA   #$01      ; 6522PB0输出为1
        STA   IORB
        SR    DELAY     ; 转延时子程序
        LDA   #$0       ; 停止定时器T1
        STA   ACR
        STA   IORB      ; 6522的PB0输出为0
        LDY   #$01
        JSR   DELAY     ; 点划之间的间隙时间为一个单
                          位时间
        DEC   COUNT
        BNE   NEXT      ; 八位未完就再重复下去
FINISH  LDY   #$02
        JSR   DELAY     ; 事先已有一个单位时间的延
                          时，故这里只需要加二个单位
                          时间延时
EXIT    RTS
DELAY   TYA
        ASL   A
        ASL   A
        TAY             ; 将Y的值变大为四倍
D3      LDA   SPEED
D2      #$FA
D1      DEX
        BNE   D1
        SEC
        SBC   #$01
```

```
        BNE   D2
        DEY
        BNE   D3
        RTS
SPACE   LDY   #$7
        JSR   DELAY  RTS
TABLE   BYTE    $73, $31, $55, $32, $3F, $2F
                $27, $23, $21, $20, $30, $38
                $3C, $3E, $01, $01, $01, $0
                $01, $4C, $01, $05, $18, $11
                $0C, $02, $12, $0E, $10, $0
                $17, $0D, $14, $07, $06, $0
                $16, $1D, $0A, $08, $03, $0
                $11, $0B, $19, $1B, $1C
```

程序比较难理解的地方是：

```
    LDA  TABLE-$2C, X
```

这一条指令如何寻址，为什么它可以找到表5—2中所列出的转换码？为此我们首先看看表5—2。表中头一个字符的ASCII码是＄2C，以后基本上是按顺序加1。但中间有几处地方ASCII码的值是间断的，我们在程序中已安排一个表。即从地址TABLE开始，顺序存放表5—2中的转换码。但要注意，凡是表5—2中ASCII码值中断的地方我们应补上**01**，因此这张表就等于包含了从ASCII码＄2C到＄5A的全部转换码。取**01**征是因为这个值不会发出任何声音，填补了这些单元以后，我们就可以方便地用绝对变址的方法查表了。具体办法是：我们把ASCII码征作为移量，而基址则用表格的首址减去＄2C。例如“，”的ABCII码为＄2C，所以按上述办法真地址即

基址＋偏移══TABLE－＄2C＋＄2C══TABLE

其它的字符亦可依此类推。

下面我们介绍频选电话的例子。一般电话都是用拨号转盘产生脉冲的方法来完成数字到电信号之间的转换，我们这里介绍的是另一类电话，这种电话是用按钮而不用拨号转盘。电话机上有一个数字按钮盘，如图5—9所示。每按下一个按钮就产生二个音频信号，一个是高音，一个是低音。图中虚线框内即它们的频率。

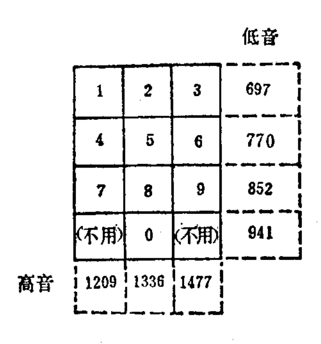


图5—9 按钮式电话频率表(美国)

如按下1钮，即产生1209赫和697赫两个音频信号。这种电话是用不同频率所组合的音频信号来完成数值与电信号之间的转换。而电话的接收部分则依照频率的组合情况来决定所呼叫电话号码，所以称它为频选（频率选择）电话。我国目前公用电话尚未使用频选的方法，但是在有些企业或部门内部已有使用频选电话系统的。这些频率是经过精心挑选的，尽可能减小高次谐波的串扰，又要尽可能少占频带宽度，因此要求频率有相当的准确性，一般这都是由音叉振荡器或高品质的LC电路来完成。现在我们这里向读者介绍一种用单板计算机完成的办法。为了演示你也可以在输出部分接一个喇叭，这里用两个 6522 的定时器1来产生两种不同频率的振荡，程序的目的在于把你所要求呼叫的电话号码转换成所对应的音频组合信号。我们在计算机中保持着频率表，它确定了每个数字所要求产生的两个音频信号。显然和上面的例子相同，每个音频信号对应着往定时器1里写入两个八位二进制数字，因此一个数字要在存贮器的频率表中占用四个存贮单元，因此用数字来进行变址寻址时要求乘4。程序开始进行前要求把电话号码数字

以十六进制的形式存放在存贮器中，每个数字用一个单元，而且0F表示电话号码结束，例如656531即应在存贮单元中顺序存放 06 05060503010F，占用七个存贮单元。本程序可适应任何长度的电话号码。

在说明程序之前我们先来计算一下每种频率所对应的常数。例如 $f=697\text{Hz}$，$T=\frac{1}{f}=1434.7$微秒。如果计算机系统时钟为 1 兆赫，则$N=717$，但实际上精确地说6522定时器产生的半周期比

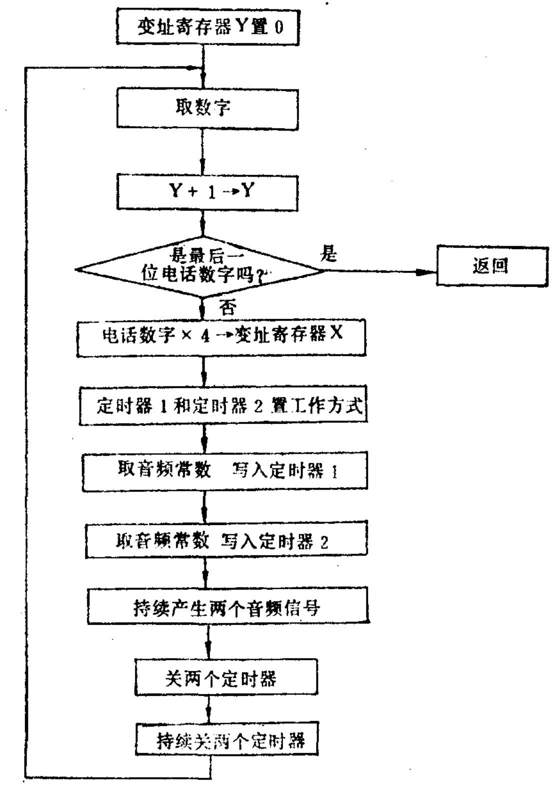


图5—10 频选电话框图

对N大1.5左右，因此实际上取$N=716$。表5—3列出了全部频率所应的定时器常数。

图5—10是程序框图。

表5—3　　频率与定时常数

频　率	半周期(微秒)	定时器常数	定时器常数（十六进制）
697	717.4	716	02CC
770	649.3	648	0288
852	586.8	585	0249
941	531.3	530	0212
1209	413.5	412	019C
1336	374.5	372	0174
1477	338.5	337	0151

```
PHONE   LDY   #$0            ; 为取电话号码数字，置变
                               址寄存器Y为零
DIGIT   LDA   (NUMPTR),Y     ; 取电话号码数字
        INY
        CMP   #$0F           ; 查看是否为电话号码结束
                               标记。
        BNE   NOEND          ; 不是结束，应跳转到发音
                               频信号程序
        RTS
NOEND   ASL   A              ; 将数字乘以4
        ASL   A
        TAX                  ; 将数字送到变址寄存器X
```

```
        LDA   #$C0          ；初始化6522的定时器为连
                             续运行方式，PB7输出
        STA   ACR1
        STA   ACR2
        LDA   TABLE,X       ；从音频常数表中取低音信
                             号的低八位常数
        STA   T1L-L         ；写入定时器1
        INX
        LDA   TABLE, X      ；取低音信号的高八位常数
        STA   T1L-H
        STA   T1C-H         ；起动低音信号
        INX
        LDA   TABLE, X      ；取高音信号低八位常数
        STA   T2L-L         ；写入另一片6522的定时器1
        INX
        LDA   TABLE, X      ；取高音信号的高八位常数
        STA   T2L-H
        STA   T2C-H         ；起动高音信号
        LDX   #ONDUR        ；取发音持续时间常数
ON      JSR   DELAY
        DEX
        BNE   ON
        LDA   #$0           ；停止两个定时器
        STA   ACR1
        STA   ACR2
        LDX   #OFFDUR       ；取停止持续时间常数
OFF     JSR   DELAY
        DEX
```

```
        BNE   OFF
        JMP   DIGIT              ; 再去取下一个电话号码数
                                   字
DELAY   LDA   #DELCON            ; 置延时子程序的延时常数
WAIT    SEC
        SBC   #$01
        BNE   WAIT
        RTS
TABLE   BYTE  $13, $02, $76,$01,$CD,$02, $9E
              $01, $CD, $02, $76, $01, $CD,
              $02, $53, $01, $89, $02, $9E,
              $01, $89, $02, $76, $01,$89,$02
              $53, $01, $4B, $02, $9E, $01,
              $4B, $02, $76, $01, $4B, $02,
              $53, $01
```

§5—3 APPLE Ⅱ计算机的彩色绘图与发声

APPLE 计算机是世界上首屈一指的八位个人计算机。它以6502为 MPU。它的彩色绘图有两种工作方式，其一为低分辨度；另一为高分辨度，我们这里仅就它的低分辨度彩色绘图程序的编写进行一般的讨论。APPLE机内装有一个2″的小喇叭可以发出声音，本节的末尾还将向大家介绍一个APPLE机电子琴程序。

绘图有两种情形：一是固定图形——彩色图案；另一是活动图形。首先我们来谈谈固定图形的绘制。为此我们用表5—4列出APPLE机监控程序中可调用的绘图子程序的名称及入口。低分辨度图形有十六种颜色可供选择，各种彩色所对应的数字如表5—5所示。APPLE Ⅱ 计算机屏幕工作方式由它所规定的八个软开关

表5—4　APPLE Ⅱ机MONITOR中的绘图子程序

入　口	名　称	功　能	调用前需要预置的寄存器	调用后将影响的寄存器
$F800	PLOT	在低分辨度图形第1页上画一个图形点	横坐标送Y寄存器，纵坐标送A累加器	A
$F819	HLIN	画一条低分辨度水平线	纵坐标送累加器A，水平线左端点横坐标送Y寄存器，右横坐标送$2C存贮单元	A和Y
$F828	VLIN	画一条低分辨度垂直线	横坐标送寄存器Y，垂直线上端纵坐标送累加器A，下端纵坐标送$2D存贮单元	A
$F832	CLRSCR	使整个图形屏幕(48行)为黑色	没有	A和Y
$F836	CLRTOP	使彩色屏幕中上40行黑色，保留以下的四行文本	没有	A和Y
$FB40	SETGR	设置低分辨度图形和文本(四行)混合方式，清屏幕为黑色。	没有	A和Y
$F864	SETCOL	置低分辨度图形彩色	彩色数字送累加器A	A
$FC58	HOME	清屏幕（文本方式)，并把光标放在左上角	没有	A和Y

控制。我们就将它们列在表5—6中。

如果你要选择在屏幕第1页上绘制全屏幕低分辨度图形，那么

表5—5　APPLE II机的低分辨度图形的彩色

数字(十六进制)	色彩	数字(十六进制)	色彩
0	黑色	8	棕色
1	洋红	9	桔黄色
2	深蓝	A	灰色2
3	紫色	B	粉红色
4	深绿	C	浅绿
5	灰色1	D	黄色
6	蓝色	E	海蓝色
7	浅蓝	F	白色

表5—6　APPE II屏幕软开关

地址单元(十六进制)	功能
$C050 $C051	选择图形方式 选择文本方式
$C052 $C053	选择全屏幕显示图形或文本 选择图形和文本混合方式（底部四行文本）
$C054 $C055	选择屏幕第1页（图形或文本） 选择屏幕第2页（图形或文本）
$C056 $C057	选择低分辨度图形方式 选择高分辨度图形方式

你应写四条指令：

```
        LDA   $C052     ; 全屏幕显示
        LDA   $C054     ; 屏幕第1页
        LDA   $C056     ; 低分辨度
        LDA   $C050     ; 图形方式
```

由此可以看出对于八个软开关，每一个开关控制一种选择。如果你决定选取那一种，应对它所对应的地址单元进行存取（即使用LDAD或STORE指令）。这四条指令的顺序是可以任意的，不过一般说来LDA $C050要求放在清除屏幕以后，以免使屏幕上出现乱七八糟的彩色块。

例如我们想以图形和文本混合方式在屏上划一条彩色的对角线。

```
        JSR   HOME     ; 清除全屏文本，使光标放在左上角
        JSR   SETGR    ; 置混合方式（第1页）上40行为黑色
        LDA   #$0B     ; 置颜色为粉红色
        JSR   SETCOL   ;
        LDY   #0       ; 从左上角开始画，纵坐标=横坐标=0
  LOOP  TYA
        JSR   PLOT     ; 画一个低分辨彩色方块
        INY            ; 为划下一个方框准备好坐标
        CPY   #$28     ; 是否已划到右下角?
        BNE   LOOP     ; 没有，再画
        BRK
```

现在我们介绍一个绘制彩色固定图形的一般程序。首先，我们分析一下图形的构成，任何图形都可以说是由水平线，垂直线和一些点构成的。在表5—4中已有画水平线，垂直线和画一个点的子程序。当然在调用它们之前必须将坐标写入指定的寄存器，由此我们可以设想一般的绘图程序无非是不断地调用 这 些 子 程

序，画出水平线、垂直线以及点。各种不同的图形无非是调用前的坐标不同，调用的次数不同。我们可以事先将一张彩色图片的所有水平线、垂直线和点的彩色和坐标（用透明坐标纸即可方便地确定坐标）都一一记录下来，按水平线、垂直线和点分三张表列出，然后将它们顺序存放在 APPLE 计算机用户存贮区的某个地方（如图5—11所示），表中线1、线2、点1、点2系指第一条水

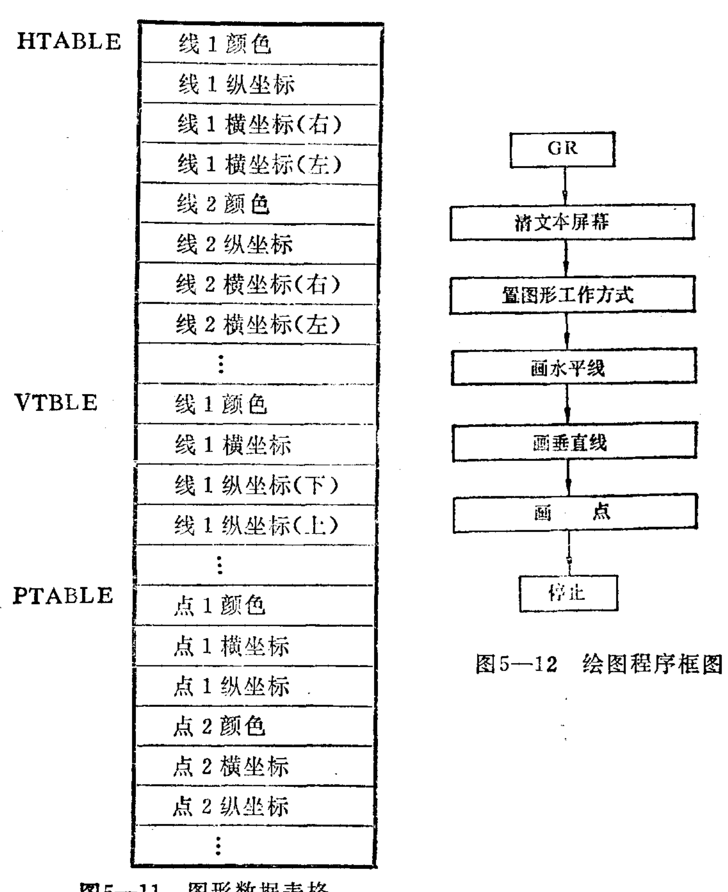


图5—11 图形数据表格

图5—12 绘图程序框图

平线（或第一条垂直线）和第一个点等等。注意坐标的列出顺序，不能随意交换。有了这一张图形数据表，我们就可以用以下列出的程序绘制其图形，图形改变，程序基本上不变，只需改变这张表格里的数据和长度。

```
GR  JSR  HOME
    JSR  SETGR       ；置混合方式
    JSR  HDRAW
    JSR  VDRAW
    JSR  PDRAW
    BRK
```

图5—12是一般绘图程序的框图，这个框图一看就懂，全部是调用子程序，和用手绘制图形的过程相似。程序如以上所示。图5—13是画水平线子程序框图。画水平线的子程序如下：

```
HDRAW   LDX  #$0
LOOP1   LDA  HTABLE,X；取水平线颜色值
        JSR  SETCOL
        INX              ；准备取表中下一个值
        LDA  HTABLE,X；取纵坐标送A
        INX
        LDY  HTABLE,X；取右横坐标送$2C
        STY  $2C
        INX
        LDY  HTABLE,X；取左横坐标送Y
        JSR  HLIN        ；画一条水平线
        INX
        CPX  #HNUM       ；HNUM = $(4×水平线条
                           数)
        BNE  LOOP1       ，没画完最后一条线，再重复
```

```
                                    下去
        RTS
```

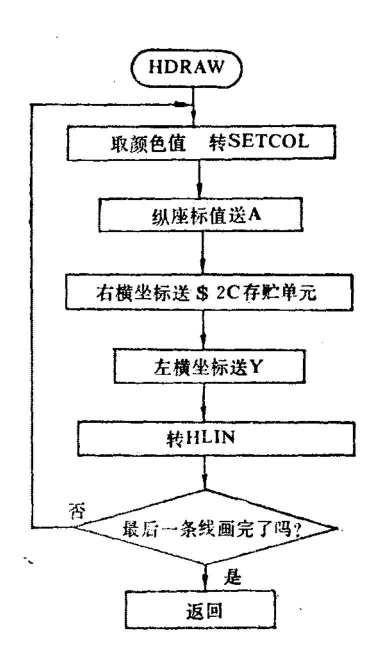


图5—13 画水平线框图

同样的道理可写出画垂直线和点的子程序。

```
VDRAW    LDX   #$0
LOOP2    LDA   VTABLE,X      ; 取垂直线颜色值
         JSR   SETCOL
         INX
         LDY   VTABLE,X      ; 取横坐标值
         INX
         LDA   VTABLE,X      ; 取纵坐标(下)
         STA   $2D
```

```
        INX
        LDA   VTABLE,X; 取纵坐标(上)
        JSR   VLIN     ; 画一条垂直线
        INX
        CPX   #VNUM    ; VNUM = $(4×垂直线条
                         数)
        BNE   LOOP2    ; 没画完最后一条线，再重复
                         下去
        RTS            ; 画完了则返回
PDRAW   LDX   #$0
LOOP3   LDA   PTABLE,X; 取点颜色值
        JSR   SETCOL   ; 置彩色
        INX
        LDY   PTABLE,X; 取横坐标
        INX
        LDA   PTABLE,X; 取纵坐标
        JSR   PLOT     ; 画一个点
        INX
        CPX   #PNUM    ; PNUM = $(3×点数)
        BNE   LOOP3    ; 没画完最后一个点，再重
                         复下去
        RTS            ; 画完了返回
```

以下我们来谈谈活动图形的绘制。图5—14给出了绘制活动的基本算法，头一个方框要求在屏幕上画一张固定的图样，显然这可以仿照上面讲过的办法。第二个方框是一段延时程序，有了它才可以让人们看清楚。一般这段时间太长会使人们看来觉得活动进行得不连惯，太短又不能让人们看清原图样，一般取0.1～0.3秒左右。擦除有两种办法：一是清除整个屏幕，然后再涂上背景

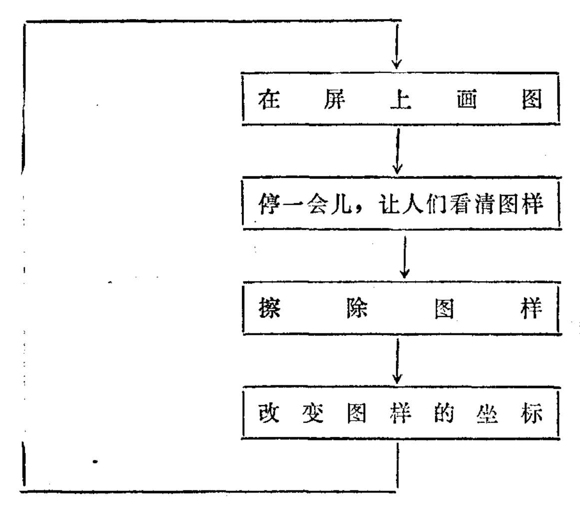


图5—14　活动图形绘制

颜色；另一是以背景颜色重画原来的图样。头一种方法会使人看到擦除过程造成的屏幕闪烁，后一种方法看上去不会闪烁，也很容易做到。只要你再次调用画图子程序，但这一次所设置的颜色

是背同。

个移动的作例子来进一步说明活动图形程

编写。

例　移动的方框

```
HLIN     EQU   $F819
VLIN     EQU   $F828
CLRSCR   EQU   $F832
SETCOL   EQU   $F864
WAIT     EQU   $FCA8
         ORG   $300
         LDA   $C052   ；选择全屏幕显示
         LDA   $C054   ；选第1页
```

```
        LDA   $C056    ; 低分辨度
        JSR   CLRSCR   ; 清屏幕
        LDA   $C054    ; 选图形方式
START   LDA   #15      ; 白色
        JSR   SETCOL
        JSR   DRAW     ; 画一个白色方框
        LDX   #03      ; 延时0.3秒
DELAY   LDA   #200
        JSR   WAIT
        DEX
        BNE   DELAY
        LDA   #0       ; 黑色——背景颜色
        JSR   SETCOL
        JSR   DRAW     ; 画一个黑色方框,即擦除
                       ;   了原来图样
        CPY   #39      ; 是否移到了屏幕的边框?
        BEQ   BREAK    ; 如果是就停下来
        INC   LHOR     ; 因为方框是水平方向移
                       ;   动,故只需改变它左右两
                       ;   端水平坐标
        INC   RHOR
        JMP   START
BREAK   LDA   $C051    ; 选文本方式
        BRK
DRAW    LDA   TVERT    ; 取方框的上边框纵坐标
        LDY   LHOR     ; 取上边框左端横坐标
        LDX   RHOR     ; 取上边框右端横坐标
        STX   $2C
```

```
         JSR   HLIN       ; 画上边框
         LDA   BVERT      ; 取下边框纵坐标
         LDY   LHOR       ; 因为子程序要修改Y，需
                            要再一次置数
         JSR   HLIN       ; 画下边框
         LDY   LHOR       ; 取左边框横坐标
         LDA   TVERT      ; 取左边框上纵坐标
         LDX   BVERT      ; 取左边框下纵坐标
         STX   $2D
         JSR   VLIN       ; 画左边框
         LDY   RHOR       ; 取右边框横坐标
         LDA   TVERT      ; 因为VLIN会修改A值，再
                            一次置右边框上纵坐标
         JSR   VLIN
         RTS
LHOR     DFB   0
RHOR     DFB   6
TVERT    DFB   20
BVERT    DFB   25
```

最后我们APPLE Ⅱ 的发声与电子琴程序。在介绍程序之前，先简述一下APPLE Ⅱ发声的物理原理。在其内部有一个小喇叭，它受一个接成计数方式工作的D 触发器的驱动，为了使喇叭

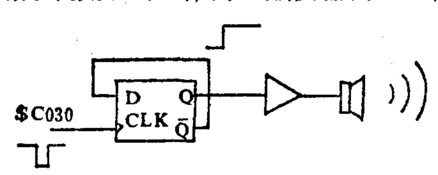


图5—15　APPLEII机内喇叭部分电路

声出的音量大一些，加入了一个简单的放大电路。当触发器不断翻转时，通过喇叭的电流也跟着变化。触发器的地址是＄C030，也就是说当MPU对地址＄C030进行读或写时，触发器的CLK(时钟)端将产生一个脉冲使触发器翻转，如果我们不断地访问地址＄C030，触发器将不断地翻。如：

```
CLICK   STA   $C030
        JMP   CLICK
```

程序运行后就可不停地对地址＄C030进行写操作，触发器的CLK端和Q端将有如图5—16的波形。CLK的周期为7μs是由于头一条指令执行时间是4μs，后一条指令则是3μs。Q端输出信号的周期为14μs，频率是71429赫，这个频率大大超过了人耳的可闻阈。但是如果我们在两条指令之间加入适当的延时，就使Q端输出信号的频率达到所要求的数值。在第二章表2—1中我们列出了子程序WAIT的延迟时间是：

$$T = \frac{1}{2}(26 + 27A + 5A^2)\mu s$$

式中A是累加器的值。表5—7列出了采用WAIT子程序发出钢琴的中音C调到低音C调一共十四个音调的频率以及所要求的A值。

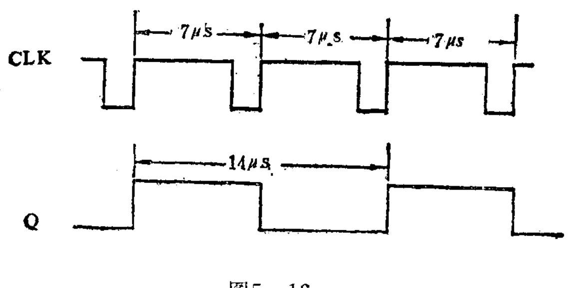


图5—16

表5—7　　　　　　频率与累加器A值

中音	频率	A值(十六进制)	低音	频率	A值(十六进制)
C	261.7	13	C	130.8	26
D	293.7	12	D	146.8	24
E	329.6	11	E	164.8	22
F	349.2	10	F	174.6	20
G	392.0	0F	G	196.0	1E
A	440.0	0E	A	220.0	1C
B	493.9	0D	B	246.9	1A

现在我们来介绍一个简单的APPLE机电子琴程序。我们定义APPLE键盘上的A、B、C、D、E、F、G分别与低八度的音调相对应，而CTRL-A～CTRL-F键则分别与其中音的A～G音调相对应。这样当我们按下一个键时，喇叭里将发出相应的声音，所以可以用它来演奏歌曲。第二章我们提到过APPLE键盘的每个按键所对应的ASCII码，由那里可知A～G所对应的ASCII码分别是$C1～$C7，CTRL-A～CTRL-G所对应的ASCII码分别是$81～$87。

```
KEYIN  EQU  $FD1B
WAIT   EQU  $FCA8
       ORG  $300
MUSIC JSR KEYIN          ；读入所按下键的 ASCII码存放
                           在累加器A中
      CMP #$C1           ；是低音吗？
      BCC HIOCT          ；小于$C1跳转到中音去
      CMP #$C8           ；大于$C7是按错了键，应重来
```

```
       BCS MUSIC
       AND #$07          ; 计算变址值
       JMP INDEX
HIOCT  CMP #$81          ; 小于$81是按错键，应重来
       BCC MUSIC
       CMP #$88          ; 大于$87，同时又小于$C1也
                           应重来
       BCS MUSIC
       AND #$07
       ADC #$07          ; 计算变址值
INDEX  TAX
       LDA  DLYTBL-1, X; 从表中取出所按键对应
                           的A值
       TAY               ; 将A值暂存在Y中
PERIOD JSR WAIT          ; 转延迟子程序
       STA  $C030        ; 产生CLK信号,使触发器翻转
       BIT  $C000        ; 测试有新的键被按吗?
       BMI MUSIC         ; 若有,则重新回到程序的开始
       TYA               ; 若没有则继续发原来的音调
       JMP  PERIOD
DLYTBL  DFB  $0E, $0D, $13, $12, $11,  $10,
             $0F
        DFB $1C, $1A, $26, $24,$22,$20,$1E
```

注意,这里所列出的A值，同表5—7中相同,但在实际中可能要略有修改，这是因为在发声程序段中除了有调用 WAIT子程序之外，又加入了四条其他的指令，它们也要一些执行时间，因此WAIT所延迟的时间要减少一些才能使触发器Q端信号频率符合要求，故A值都需要适当地调整。

APPLE 电子琴程序框图如图5—17所示。程序中的数据表是按A～G和CTRL-A～CTRL-G英文字母的顺序将其所对应的A值存放。由于它们所对应的ASCII码分别是＄C1～＄C7和＄81～＄87，因此只需将高四位二进制数换成零就成了＄1～＄7，如果把

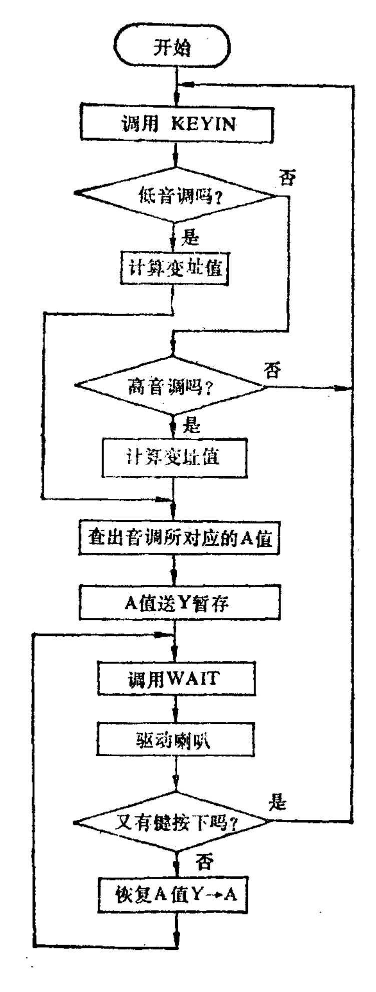


图5—17 APPLE 电子琴程序框图

这个值送入变址寄存器X，对于A～G可采用DLYTBL-1作为基址进行绝对变址寻址，即可找到对应的A值。而对于CTRL-A～CTRL-G则需要将X值加7以后再以DLYBTL-1 为基址，进行绝对变址寻址。程序中用指令BIT＄C000 来测试是否又有新的键被按下。＄C000是APPLE Ⅱ 的键盘输入的单元。当有键被按动时，这个单元的Bit7将为1电平。所以程序中用了指令BMI来判定Bit7是不是1；如果是，则重新回到程序的开始。

§5—4　电子时钟

这里介绍一个用AIM-65单板计算机完成的电子时钟。它的时钟部分主要由硬件——6522的定时器1，以及中断服务程序来完成，因此它独立于主程序之外。定时器需要服务时，向MPU请求中断，MPU暂时放下当前正在进行的工作，响应定时器中断请求，并转入中断服务程序，处理完成之后，又恢复原来被打断的工作。显然这里列出的主程序，进行一些修改，就可以使这个程序成为定时顺序控制器程序。

主程序除了包括初始化定时器程序之外还有为了使你操作这个电子时钟方便而加入的程序，它是一个小小的时钟操作控制程序——也可以夸大称它是电子时钟管理程序。它用AIM-65的自定义功能键F1调用时钟程序。运行之后当即显示一个提示符——%。程序将等待你操作四个命令键（T、M、D、C）之一来控制电子时钟。这四个键的功能是：

T键：设置当前时间。要求你以“时：分：秒”的次序来设置，并且它们之间一定要用“：”冒号分隔开。时分秒都用两位数字如09：15：08，即九时十五分八秒。(不可打成9:15:8)以保证打入的一共八个字符。

M键：你可以打入一条文字，即一个便条，但最多不能超过

十二个字符（包括空格和标点符号）。这一条文字将在显示的时间之前显示。如果你的便条不够十二个字符，你在末尾可以用回车符（RETURN）结束。如"I'LL GO BACK"被打入，当显示时间的时候，AIM-65的显示器（最多20个字符）将显示：I'LL GO BACK 12:00:00。

D键：使你用M 键所打入的便条和当时的时间立即被显示和打印。

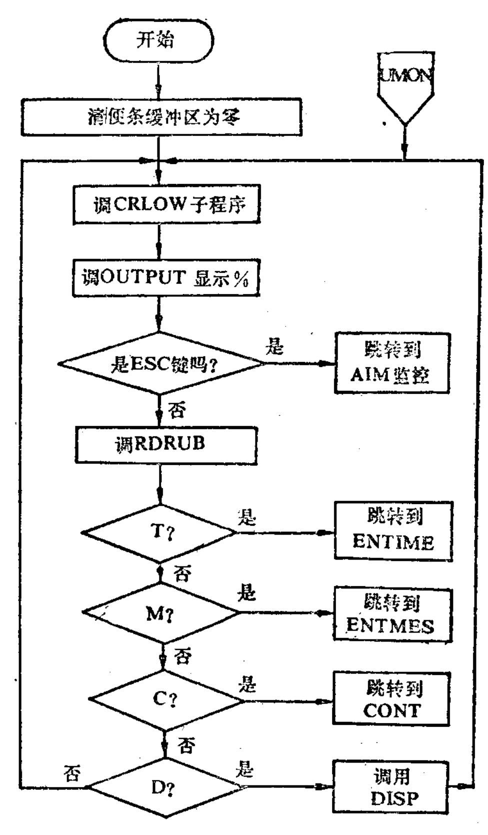


图5—18　电子时钟主程序框图

C键：使文字和时间连续不断地显示（但不打印），直到压下ESC键，即停止显示但时钟并不停止显示，而显示出提示符%号。

显示%号提示符之后，如果按下ESC键，即退出电子时钟而返回到AIM-65监控程序。图5—18是主程序框图。主程序清单如下：

```
        CRLOW = $EA13
        GETK2 = $EC82
        OUTPUT = $E97A
        RCHEK = $E907
        RDRUB = $E95F
        ROONEK = $ECEF
        T1CH = $A005
        T1LL = $A004
        ACR = $A00B
        IER = $A00E
        PRIFLG = $A411
        HRS10 = $A5
        HRS = $A6
        MINS10 = $A8
        MINS = $A9
        SECS10 = $AB
        SECS = $AC
        INTCNT = $98
        BUFFER = $99
        TSECS = $96
; SET UP F1 KEY         ; 建立功能键F1，使$10C、$10D
        * = $10C          和$ 10E 三个存贮单元中放三个
        JMP START         字节4C，00，02十六进制码
; SET UP IRQ VECTOR; 建立中断向量，即在 $A400和
```

```
        *=$A400          $A401两存贮单元中放中断服
        *WOR INT         务程序入口地址(低字节在
                         $A400)
; PROGRAM START LOC.
        *=$200
START   LDX #00
        LDA #$20
        JMP PAD         ; 清便条缓冲区
UMON    JSR CRLOW
        LDA #$'%         提示符%显示
        JSR OUTPUT
        JSR RCHEK        转查一下是不是ESC键
        JSR RDRUB        ; 转读键盘输入
        JSR CRLOW        ; 回车换行
        LDX #00
        CMP #$'T         ; 是T命令吗?
        BNE MTEST
        JMP ENTIME
MTEST   CMP #$'M         ; 是M键吗?
        BNE CTEST
        JMP ENTMES
CTEST   CMP #$'C         ; 是C命令吗?
        BNE DTEST
        JMP CONT
DTEST   CMP #$'D         ; 是D命令吗?
        BNE UMON
        JSR DISP
        JMP UMON
```

```
ENTIME LDA TIMSG,X   ; 显示 ENTER TIME:
       JSR OUTPUT
       INX
       CPX #12
       BNE ENTIME
       LDX #0        ; 打入时间
TINLP  JSR RDRUB     ; HH:MM:SS
       STA HRS10,X
       INX
       CPX #08
       BNE TINLP
       LDA #$10
       STA INTCNT;(16)D→INTCNT (内部计数器)
       LDA #$C0      ; IER = $C0, 允许T1中断请求
       STA IER
       LDA #$40      ; 定时器1连续中断单稳方式, 详见
       STA ACR         § 4-2
       CLI           ; 清中断禁止标志
       LDA #$22
       STA T1L-L
       LDA #$F4
       STA T1C-H     ; 使定时器1每1/16秒产生一次中断
       JMP UMON
ENTMES LDA MESG,X    ; 显示 "MESSAGE:"字样
       JSR OUTPUT
       INX
       CPX #08
       BNE ENTMES
```

```
        LDX #0
MINLP JSR RDRUB
        CMP #$0A          ; $0A是换行符的ASCII码
                          查是否为换行符
        BEQ MINLP         ; 是，再重取下一个字符
        CMP #$0D          ; 是回车符吗?
        BNE NOTCR         ; 不是，则跳转到NOTCR
        LDA #$20          ; 如是则以空格符填满缓冲区
PAD     STA BUFFER, X
        INX
        CPX #12
        BNE PAD
        JMP UMON
NOTCR STA BUFFER,X ; 将打入的字符写入缓冲区
        INX
        CPX #12
        BNE MINLP
        JMP UMON
DISP    LDA BUFFER,X  ; 显示便条及时间
        JSR OUTPUT
        INX
        CPX #20
        BNE DISP
        RTS
CONT    LDA #0
        STA PRIFLG        ; 关打印机
CONTLP JSR CRLOW
        LDX #0
```

```
        JSR  DISP        ; 显示便条和时间
        LDA  SECS
        STA  TSECS       ; 将显示的秒、数、存入TSECS单元
DISLP   LDA  SECS
        CMP  TSECS       ; 中断服务程序修改 SECS 单元，
                           现在是查看是否已经修改
        BEQ  DISLP       ; 若尚未改就继续等待
        JSR  ROONEK      ; 查看是否有键按下？
        DEY              ; 如果没有键压下Y=0，若有键压
                         下Y>0
        BMI  CONTLP
        LDX  #0          ; 如果有，查看是ESC 键吗？
        JSR  GETK2
        CMP  #$1B        ; 是 ESC 键被按下吗？ $1B 是
                           ESC键的ASCII码
        BNE  CONTLP      ; 不是，返回CONTLP
        LDA  #$80        ; 是，则重新开打印机
        STA  PRIFLG
        JMP  UMON
TIMSG· BYT 'ENTER TIME':
MESG·  BYT 'MESSAGE:'
; INTERRUPUT  ROUTINE     以下为中断服务程序。
; ASCII  CODES  FOR       时、分、秒的ASCII 码分别存
; HOURS  IN  $A5&$A6      放在$A5，$A6和 $A8、$
; MINS  IN  $A8&$A9       A9以及$AB，$AC 存贮单
; SECS  IN  $AB&$AC       元
    INT PHA
        TXA
```

```
PHA                ；保存累加器A和寄存器X
DEC  INTCNT        ；修改内部计数器的值(减1)
BNE  TIMOK         ；不等于零，跳到TIMOK
LDA  #16           ；是零，则使内部计数器恢复
STA  INTCNT
LDA  #$30          ；$30是0的ASCII码
INC  SECS          ；秒单元内容加1
LDX  #$':          ；“:”的ASCII码为$3A
CPX  SECS          ；(SECS)为$3A吗？
                   即是否已加过拾次
BNE  TIMOK
STA  SECS          ；将$30(即0的ASCII码)存放在
                    SECS单元
INC  SECS10        ；拾秒存贮单元加1
LDX  #$'6          ；6的ASCII码是$36
CPX  SECS10        ；查其中是6的ASCII码吗？
BNE  TIMOK         ；不是，转TIMOK
STA  SECS10        ；将$30存放在SECS10单元中。
INC  MINS          ；分存贮单元加1
LDX  #$':          ；查其中是$3A吗？
                    即是否加过拾次
CPX  MINS
BNE  TIMOK         ；不是，转TIMOK
STA  MINS          ；将$30存入MINS单元
INC  MINS10        ；拾分单元加1
LDX  #$'6          ；查拾分单元是6的ASCII 码吗？
CPX  MINS10
BNE  TIMOK         ；不是，转TIMOK
```

```
        STA MINS10
        INC HRS          ; 时存贮单元加1
        LDX #$'：
        CPX HRS          ; 查时存贮单元为拾吗?
        BNE NUDAY        ; 不是，转NUDAY
        STA HRS          ; 将$30存入HRS单元
        INC HRS10        ; 将拾时单元加1
        JMP TIMOK
NUDAY   LDX #$'4         ; 查HRS单元是4吗?
        CPX HRS
        BNE TIMOK        ; 不是，转TIMOK
        LDX #$'2         ; 查拾时单元是不是2
        CPX HRS10
        BNE TIMOK        ; 不是，转TIMOK
        STA HRS10        ; 是，新的一天开始
        STA HRS
TIMOK   LDA T1L-L        ; 清中断标志
        PLA              ; 恢复现场
        TAX
        PLA
        RTI              ; 返回主程序
        END
```

图5—23是中断服务程序框图。

T命令

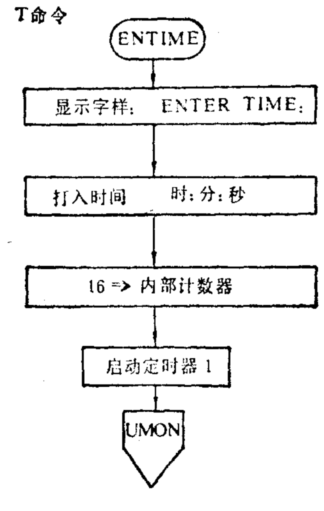


图5—19 打入时间程序框图

M命令

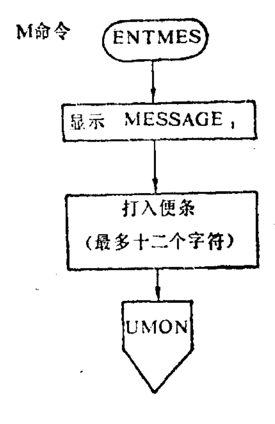


图5—20 M命令

D命令

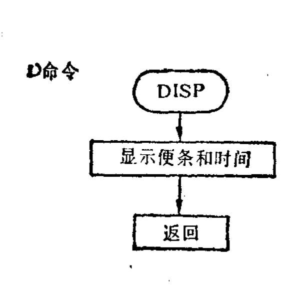


图5—21 D命令

C命令

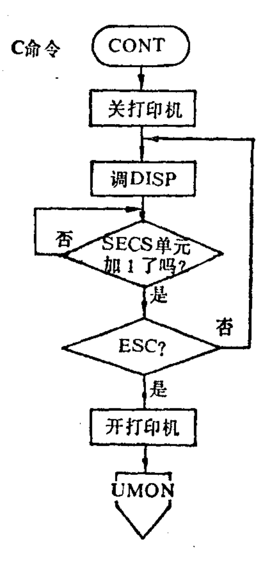


图5—22 C命令

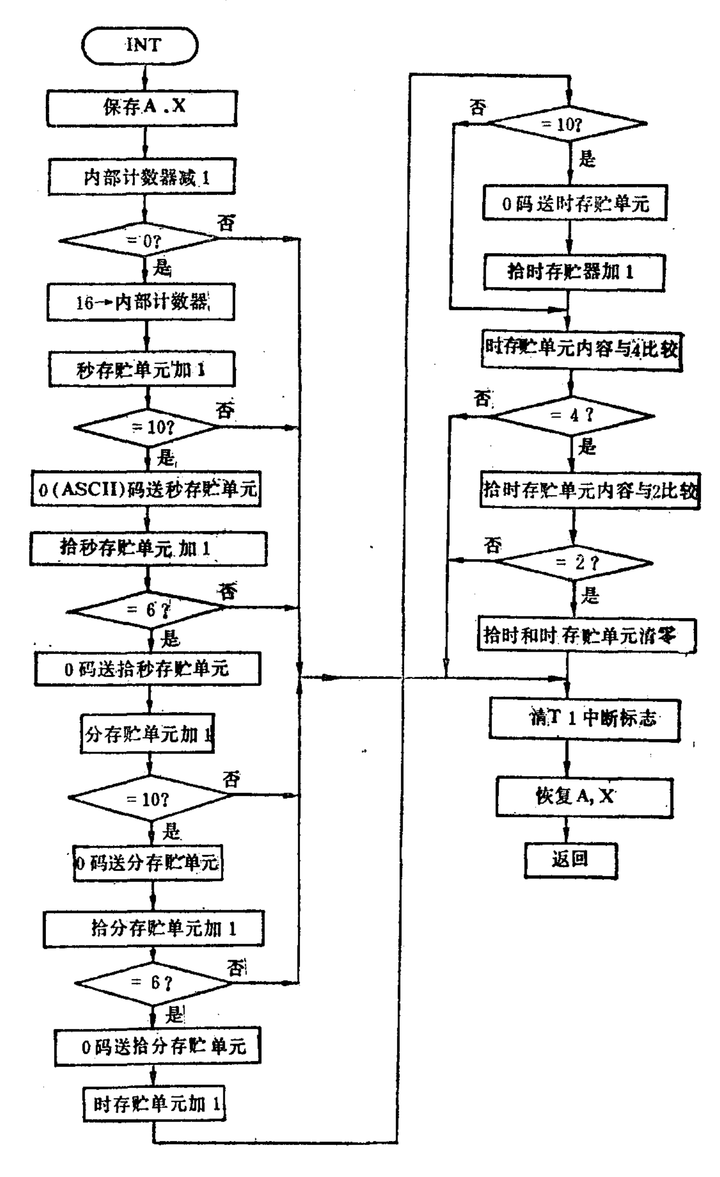


图5—23　中断服务程序框图

§5—5　数—模(D/A)与模—数(A/D)转换

本节的目的是介绍6502的单板机(如AIM-65和SYM-1) 如何应用来进行数—模与模—数转换。它们都有用户可以使用的6522芯片。可以用其中的一片来进行这项工作，如图5—24连接。图中MC1408是8bit数模转换芯片，M_1、M_2分别是运算放大器和电压比较器。有关它们的较详细的资料请参阅线性集成电路芯片手册。图中R可调。由它的大小决定着M_1的输出幅度，如R取6500欧左

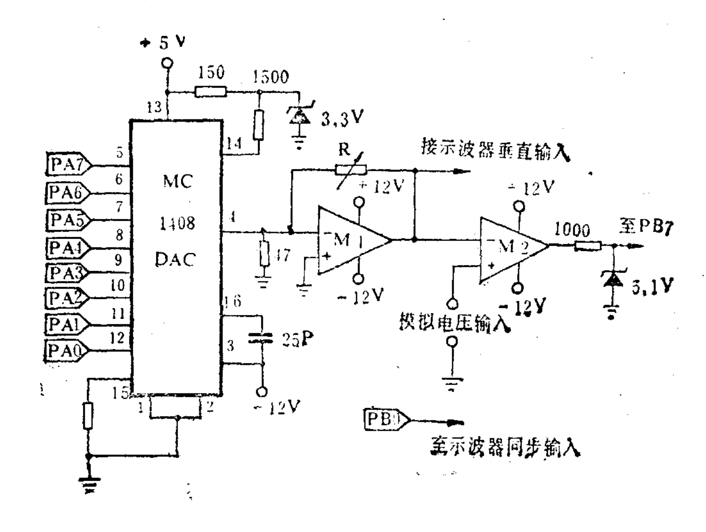


图5—24　D/A和A/D转换电路

右，M_1可输出0～+10V，即当PA0～PA7为**00000000**，M_1输出为0伏，当PA0～PA7为**11111111**，M_1输出为+10伏。M_2的输出经过一电阻送到6522B口的PB7，而B口的另一个头 PB0则输出到示波器外同步输入端。

图5—24如果配合以下的程序：

```
        LDA #$FF
        STA DDRA     ; 定义6522A口所有的线为输出线
LOOP    INC IORA     ; 使6522A口输出值加1
        JMP LOOP     ; 再返回到LOOP
```

即可使M_1的输出端产生一锯齿形电压。如果你用一示波器如图5—24接上（不必接外同步信号）适当调整示波器的扫描速率，即可使它的屏上出现一稳定的锯齿波。修改上面的程序还可使M_1输出三角波、矩形波、阶梯波等。

A口输出的值，由于执行到INC IORA指令而逐渐增加，一直到IORA的输出为11111111（全1），再加1就变成了全0，然后又逐渐上升。因此这些值经过MC1408进行转换以后就变成直线上升的电压，升到最大值后又突然下降到最小值，然后又直线上升，如此循环而形成了锯齿形的电压。显然严格地讲，M_1输出的应是台阶式线性上升的电压，而不是直线上升的电压，但如果示波器扫描速率不十分高，台阶将显不出来，而看到的却是直线上升的电压。

M_2是一个电压比较器，它的两个输入端分别接到M_1的输出和另一个模拟电压。M_2的输出端经过一个稳压电路而接到6522B口的PB7端。当M_1的输出电压大于输入的模拟电压时，M_2的输出将为负，即PB7为0；当M_1的输出电压小于模拟电压时，M_2的输出为正，PB7为1。

图5—25所给出的是用图5—24电路去测量模拟电压而构成的一个数字式电压表所应配程序的框图。

我们开始使A口输出为零，如果这时有模拟电压输入，它当然会大于M_1的输出，故PB7为1，使程序进入IORA+1→IORA的循环，一直到A口值刚刚比模拟电压大1时，程序就由于PB7为0，而转到显示A口的值，然后又从头开始。如果我们调整R的值，使IORA输出为全1时模拟电压为2.55V（或别的值亦可），使A口

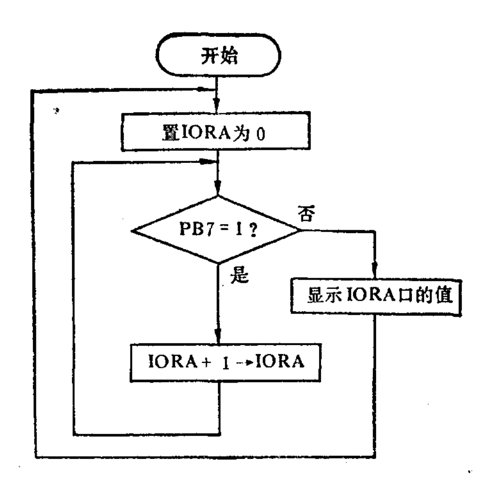


图5—25 D/A转换电压表框图

的值予以显示，图5—24电路就成了一个数字电压表了。程序如下：

```
START   LDA #$FF
        STA DDRA   ；使6522的A口所有线为输出
AGN     LDX #$0    ；使A口从零开始
RAMP    STX IORA
        LDA IORB   ；查PB7为1吗？
        BPL DISP   ；PB7为0即转显示
        INX        ；准备A口的值加1
        JMP RAMP   ；继续循环
DISP    JSR CRLF   ；这里采用的是AIM-65的监控程序中的子程序，这里可将A口的值以十六进制形式显示出来。
        JSR NUMA
        JMP AGN
```

显然，如果将A口的值转换成三位十进制数显示将会读起来更方便些，这些程序在第三章中曾经讲过。读者可将那些程序运

用到这里将A口的数值转换成三位十进制数，然后再将每一位数换成ASCII码再输出（输出对AIM-65单板计算机可调用OUTPUT子程序）。

用以上的办法完成模数转换比较慢，现在我们介绍一种较快的方法——逐次比较法。这种方法的算法同我们用天平去量度一个物体的质量情况很相似：天平测量必须有一整套砝码，在我们这里就表现为A口输出的二进制数值，A口的每一位可以为0或1，比较器就象一架天平，输出的结果0或1就表示是A口输出的数字还是被测量的模拟电压谁大，这很象天平的倾斜情况就表示了两个托盘里的东西谁重，人们依此可以决定下一步应如何选择砝码，显然在这里人们也可以依此决定下一步应该如何输出A口的数字。具体作法是：首先让A口的最高位输出为1，其余位为0，与被测模拟电压进行比较，如果比较器输出为0——表示A口的数字大于被测电压，下一步则应使最高位为0，次高位(PA6)为1，其它为0，再与被测量的电压进行比较。如果比较器输出为1——表示A口数字小于被测电压，下一步则应使最高位继续保持为1，而且使次高位（PA6）也为1。这样的办法从最高位到最低位逐位进行下去，最后A口的值将是接近等于被测电压的(误差不大于±1最低位)。这就是逐次比较模数转换的算法，图5—26表示了逐次比较法模数转换的程序框图。程序如下：

```
         LDA #$FF
         STA DDRA  ；使6522的A口所有线为输出线
         LDA #0
         STA DDRB  ；使6522B口所有线为输入线
FSTBIT   LDA #$80  ；置最高位为1
         TAY       ；让Y保存当前测试位
         STA $08   ；$08存贮单元保存当前值
NXTBIT   LDA $08
```

```
        STA  IORA   ；送当前值去数模转换
        LDX  #$20   ；这里加一个延时是为使MC1408
LP9     DEX           完成数模转换
        CPX  #$00
        BPL  LP9
        BIT  IORB   ；读B口的PB7
        BMI  SHFBIT
        TYA
        EOR  $08    ；置当前测试位为零
        STA  $08    ；保存当前值
SHFBIT  TYA
        LSR  A      ；为下一次比较右移Y
        TAY
        CMP  #00    ；到最低位吗?
        BEQ  DISP
        CLC
        ADC  $08
        STA  $08
        JMP  NXTBIT
DISP    JSR  CRLF      ；显示。以下子程序系指AIM-65
                         单板机监控中的子程序，
        LDA  $08
        JSR  NUMA
        JMP  FSTBIT
```

近年来已生产了不少专用的模数转换芯片，如ADC 0800，ADC 0808，ADC0 816等。这些芯片用硬件完成逐次比较的过程，因此不必象上面那样占用MPU的工作时间。这些芯片将数模转换、比较器以及多路模拟开关都集中于一个芯片，使用起

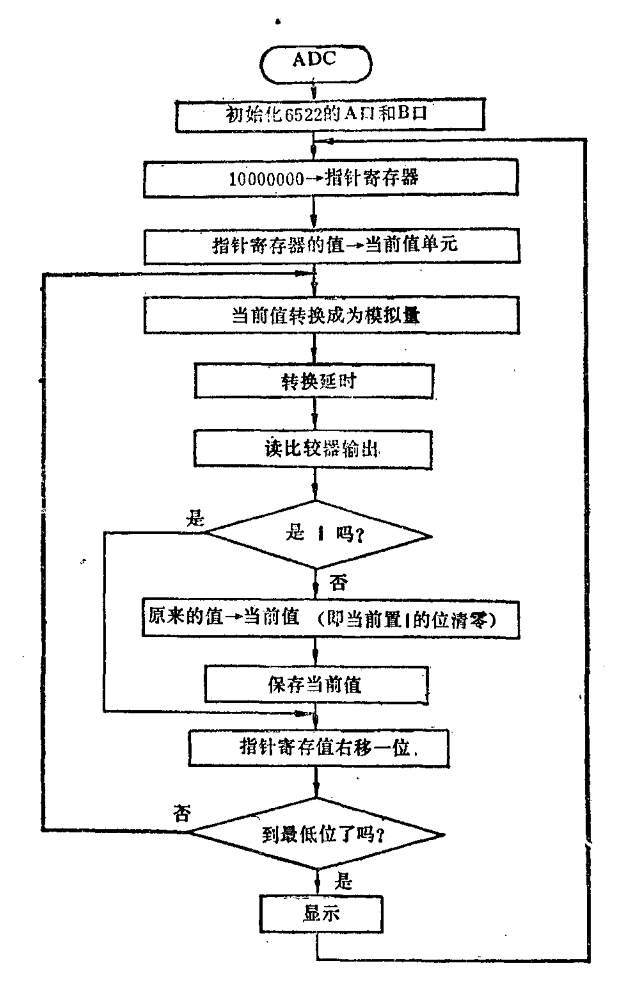


图5—26 A/D 转换框图

来要较图 5—24 的办法方便多了。以下我们介绍使用 ADC0809 (8bit8路模数转换器) 与 SYM—1配合进行多路模数转换的实例。

首先简要介绍一下ADC0809的引腿及使用方法 (详细资料请

查阅线性集成电路芯片手册)。芯片引腿图：

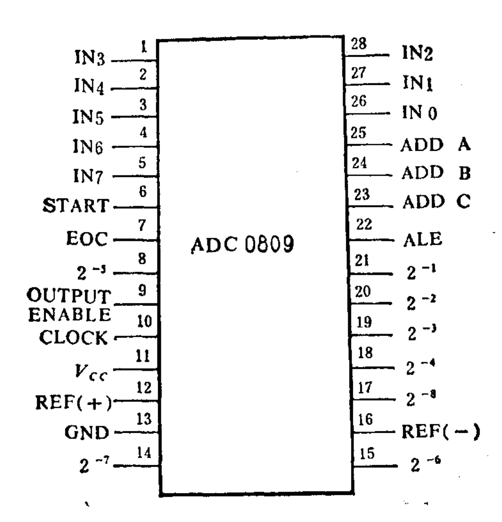


图5—27　ADC 0809引腿图

表5—8　　　　模拟电压通路地址译码表

被选模拟通路	ADD　C	ADD　B	ADD　A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

芯片使用说明：

ADC 0809 是一个八路模数转换芯片。转换结果为八位数字量，转换时间最短是100μs。八路模拟电压分别由引腿IN0～IN 7接入，这八路模拟信号对应的三位地址码接入ADD A，ADD B，ADD C三条引腿则可通过芯片内部的地址译码器和多路开关选通八路中的一路进行A/D转换。地址译码和八路模拟电压的对应关系见表5—8，当接好待转换的各路模拟电压并送入地址码之后，应向ADC0809的START(启动)引腿送入启动脉冲和向ALE（地址锁存允许）引腿送入地址锁存脉冲，ADC0809接到这两个信号后就开始A/D转换，当转换结束时，由EOC（转换结束）引腿送出高电平作为转换结束信号送到计算机，然后由计算机向ADC0809的OUTPUT ENABLE（输出允许）引腿送入一个脉冲作为输出

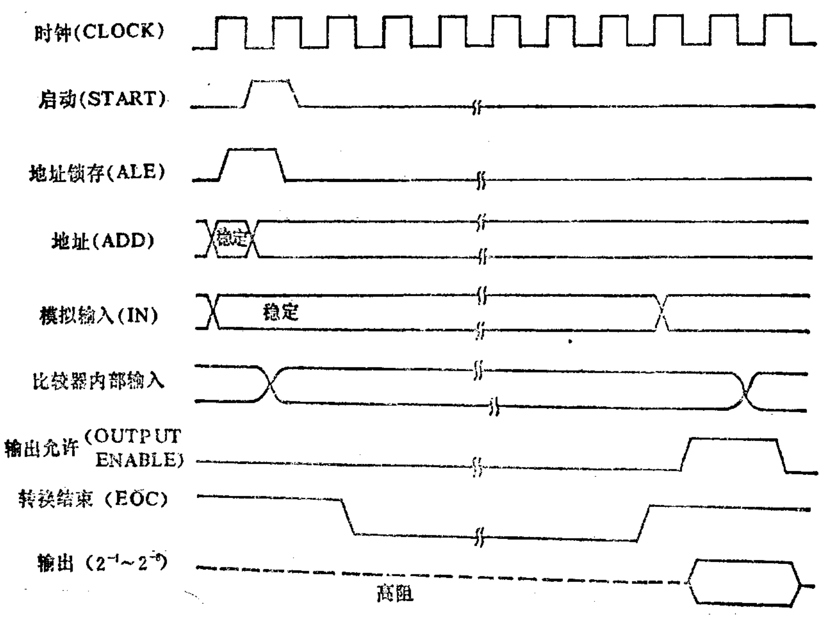


图5—28　ADC0809转换时序图

允许信号，ADC0809得到此信号时就打开芯片内部的三态输出锁存器，将A/D转换结果通过2^{-1}～2^{-8}这八条引腿送出来（2^{-1}为最高位）。

ADC0809工作时还需要供给它一个时钟信号，这由CLOCK引腿接入一个频率≤640kHz的脉冲源来实现。

电源电压V_{cc}和GND之间接入+5V。

参考电压REF(+）和REF(−）之间亦可接入+5V，REF(−）和GND接通，这样当模拟电压输入为+5V时，ADC0809输出可达FF(十六进制)，而模拟电压输入为0V时，输出为00。若是模拟电压是一个随时间快速变化的量，在A/D转换期间不能维持恒定，则在ADC0809和模拟电压之间应加入采样保持器芯片。

ADC0809的A/D转换时序如图5—28所示。

下面我们来介绍如何将ADC0809接入SYM-1单板机。将ADC0809通过SYM-1上的用户6522 VIA接口芯片（U28）接入SYM-1是很方便的。接线图详见图5—29。

SYM-1的用户6522VIA（U28）中与PIO部分有关的地址码如下：

A800　　B口数据寄存器ORB

A801　　A口数据寄存器ORA

A802　　B口方向寄存器DDRB

A803　　A口方向寄存器DDRA

A80B　　辅助控制寄存器ACR

A80C　　控制寄存器PCR

A80D　　中断标志寄存器IFR

A80E　　中断允许寄存器IER

为了简单起见，我们把ADC0809的八路模拟输入电压都接在一个由+5V直流及10k电位器构成的分压器上，调节电位器则输入的模拟电压可在0V～5V之间变化。这八路输入模拟电压的地

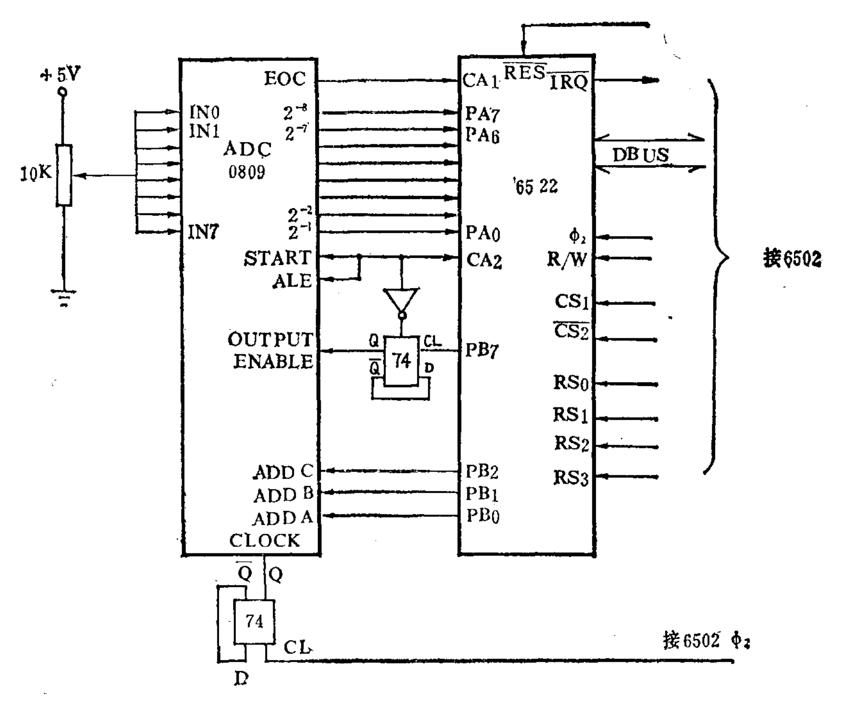


图5—29　ADC0809接入SYM—1的实验线路

址码由6522的B口低三位输出数据进行控制。START和ALE信号由6522的CA2提供。当A/D转换完毕时EOC给出高电平，6502查询到EOC有效信号到来时，就由6522的PB7送出一个脉冲并经74LS74芯片分频后送到OUTPUT ENABLE端（74由CA_2清0），接着就可由2^{-1}～2^{-8}输出端取得输出数据，经6522的A口再送往6502。为了保证ADC0809的正常工作，在其CLOCK端接入一个由6502的ϕ_2分频一次后的时钟信号（500kHz）。

按以上所述，用查询方式工作的程序如下：

指令地址	指令机器码		符号指令
0200	A9	00	LDA #00
0202	85	00	STA $00 ；0号单元清0
0204	A0	00	LDY #00 ；Y寄存器清0
0206	A9	FF	LDA #$FF ;置6522B口为输出方式
0208	8D	02 A8	STA $A802
020B	A9	00	LDA #00 ；置6522A口为输入方式
020D	8D	03 A8	STA $A803
0210	8D	0B A8	STA $A80B; A口输入不锁存
0213	A9	7F	LDA #$7F
0215	8D	0E A8	STA $A80E; 禁止各中断位
0218	A5	00	LDA $00
021A	8D	00 A8	STA $A800; 送模拟电压地址码
021D	E6	00	INC $00 ；准备好下次地址码
021F	A9	0D	LDA #$0D ;使CA2电平变化，产生⎍
0221	8D	0C A8	STA $A80C;并定义 CA1正跳沿有效
0224	A9	0F	LDA #$0F
0226	8D	0C A8	STA $A80C
0229	A9	0D	LDA #$0D
022B	8D	0C A8	STA $A80C
022E	AD	0D A8	LDA $A80D ；查询CA1
0231	29	02	AND #02
0233	F0	F9	BEQ $022E
0235	A9	00	LDA #00 ；由PB7送出⎍到

```
0237   8D   00 A8   STA   $A800      ;OUTPUT ENABLE
023A   A9   80      LDA   #$80
023C   8D   00 A8   STA   $A800
023F   A9   00      LDA   #00
0241   8D   00 A8   STA   $A800
0244   EA           NOP
0245   EA           NOP
0246   AD   01 A8   LDA   $A801     ；由A口取转换结果
0249   99   00 03   STA   $0300,Y； 送入结果单元
024C   C8           INY             ；准备下一路结果单
                                      元地址
024D   C0   08      CPY   #$08      ；八路转换完否?
024F   D0   C7      BNE   $0218     ；未完，继续转换
0251   00           BRK             ;八路都转换完,结束
```

该程序执行完毕，八路模拟电压转换所得到的数字量保存在0300～0307单元中。

下面再介绍用中断方式工作的程序，接线方法同上（图5-29所示）。

ADC0809的转换结束信号EOC通过6522的CA1向6502提供中断请求$\overline{IRQ}$，在中断服务程序中由6522的PB7向ADC0809发出OUTPUT ENABLE信号，接着就可将转换结果取到6522A口再送到存贮器中保存。每一路转换完毕都将中断一次，八路转换都处理完毕时，在SYM-1显示器上显示end字样。

主程序的内容是对6522初始化和对ADC0809给出地址码并发启动信号和地址锁存信号。

SYM-1存放中断向量的指定单元是A67E和A67F两地址，在运行程序前要用SD键将中断服务程序入口地址0250的低字节50送入A67E单元，高字节02送入A67F单元。

现将程序列出如下：

主程序

指令地址	指令操作码		符号指令		
0200	A9	00	LDA	#00	；00单元存放八路模拟电压的地址码
0202	85	00	STA	$00	
0204	A0	00	LDY	#00	；Y＝0
0206	A9	FF	LDA	#$FF	；置6522B口为输出方式
0208	8D	02 A8	STA	$A802	
020B	A9	00	LDA	#00	；置6522 A口为输入方式
020D	8D	03 A8	STA	$A803	
0210	8D	0B A8	STA	$A80B	；A口输入不锁存
0213	A9	7D	LDA	#$7D	
0215	8D	0E A8	STA	$A80E	；禁止CA1以外的中断
0218	A9	82	LDA	#$82	
021A	8D	0E A8	STA	$A80E	；允许CA1中断
021D	58		CLI		；开中断
021E	A5	00	LDA	$00	
0220	8D	00 A8	STA	$A800	；送模拟电压地址码
0223	E6	00	INC	$00	；准备好下次地址码
0225	A9	0D	LDA	#$0D	；使CA2电平变化，产生⎍
0227	8D	0C A8	STA	$A80C	；并定义CA1正跳沿有效
022A	A9	0F	LDA	#$0F	

```
022C     8D   0C A8   STA   $A80C
022F     A9   0D      LDA   #$0D
0231     8D   0C A8   STA   $A80C
0234     20   00 04   JSR   $0400；延时，在延时中等
                                   待中断
0237     4C   1E 02   JMP   $021E；开始下一路A/D转
                                   换
```

延时子程序（约2.5ms）：

```
0400    8A          TXA
0401    48          PHA
0402    A2   FF     LDX   #$FF
0404    CA          DEX
0405    D0   FD     BNE   $0404
0407    A2   FF     LDX   #$FF
0409    CA          DEX
040A    D0   FD     BNE   $0409
040C    68          PLA
040D    AA          TAX
040E    60          RTS
```

中断服务程序：

```
0250    A9   00         LDA   #00      ；由PB7送出⊓到
0252    8D   00   A8    STA   $A800    ；OUTPUT ENABLE
0255    A9   80         LDA   #$80
0257    8D   00   A8    STA   $A800
025A    A9   00         LDA   #00
025C    8D   00   A8    STA   $A800
025F    EA              NOP
0260    EA              NOP
```

```
0261   AD 01 A8   LDA  $A801 ；由A口取转换
                              结果
0264   99 00 03   STA  $0300,Y；送入结果单元
0267   C8         INY         ；准备下一路结果
                               单元地址
0268   C0 08      CPY  #$08   ；八路转换完否
026A   F0 02      BEQ  $026E  ；转换完，转026E
026C   58         CLI         ；未转换完，开中断
026D   40         RTI         ；返回
026E   A9 FF      LDA  #$FF   ；置6532A口为输
                                出方式
0270   8D 01 A4   STA  $A401
0273   8D 03 A4   STA  $A403  ；B口亦为输出方式
0276   A9 7B      LDA  #$7B   ；通过A口送
0278   8D 00 A4   STA  $A400  ；e的七段码→显示器
027B   A2 00      LDX  #00    ；使第一块显示器亮
027D   8E 02 A4   STX  $A402
0280   20 00 04   JSR  $0400  ；延时
0283   A9 37      LDA  #$37   ；通过A口送
0285   8D 00 A4   STA  $A400  ；n的七段码→显示器
0288   E8         INX
0289   8E 02 A4   STX  $A402  ；使第二块显示器亮
028C   20 00 04   JSR  $0400  ；延时
028F   A9 5E      LDA  #$5E   ；通过A口送
0291   8D 00 A4   STA  $A400  ；d的七段码→显
0294   E8         INX                示器
0295   8E 02 A4   STX  $A402  ；使第三块显示器亮
0298   20 00 04   JSR  $0400  ；延时
```

029B 4C 76 02 JMP $0276 ;继续显示 end

服务程序中CLI 这条指令其实是多余的，因为在执行 RTI 中断返回指令时，会自动把响应中断时保护进栈的 P 寄存器状态恢复到P中去，那么 I 标志位也就恢复成0状态而不需再用 CLI指令来使I＝0。

由以上程序结构可知，当主程序进入延时程序子程序段时，就会响应CA1提出的中断而进入中断服务程序，在服务程序中处理A/D 转换结果后再返回被打断的延时程序。当延时程序执行完毕返回到主程序最后一条又开始下一路的A/D转换，当八路A/D转换完毕时，在显示器上显示end字样。八路转换结果保存在0300～0307单元中。应当注意在主程序中的0234处进入的延时子程序，它的延时时间必须超过ADC0809的A/D转换时间（100μs）。否则程序将不能顺利执行。

关于 SYM-1 上六块显示器的工作原理，请看 SYM-1参考手册。这六块显示器是由SYM-1上的6532（U_{27}）A口供给七段码，而哪一块显示器亮则是由74LS145（U_{37}）的输出 DS_6～DS_0来控制。上述程序要使显示器从最左边开始显示 end 就应分别把这三个字母的七段码送到显示器上，并控制使最左边开始的三块显示器依次亮出end字样即可。

6532（U_{27}）中与显示器有关的PIO地址码有：

A400	A口数据寄存器	ORA
A401	A口方向寄存器	DDRA
A402	B口数据寄存器	ORB
A403	B口方向寄存器	DDRB

以上的程序举例可以帮助我们清楚地理解ADC0809接入微型机时软硬件构成的基本方法。

附　　录

一、表1：6502指令符号
二、表2：6502指令表

表1： 6502 指 令 符 号

记忆符	英　　文	中　　文	操　　作
ADC	Add Memory to Accumlator with Carry	带进位加法	A+M+C→A
AND	"AND" Memory with Accumulator	逻辑与	A∧M→A
ASL	Shift Left One Bit (Memory or Accumulator)	左移一位	C←| 7　0 |←0
BCC	Branch on Carry Clear	进位为零跳转	C=0跳转
BCS	Branch on Carry Set	进位为1跳转	C=1跳转
BEQ	Branch on Result Zero	结果为零跳转	Z=1跳转
BIT	Test Bits in Memory with Accumulator	位测试	A∧M
BMI	Branch on Result Minus	结果为负跳转	N=1跳转
BNE	Branch on Result not Zero	结果不为零跳转	Z=0跳转
BPL	Branch on Result Plus	结果为正跳转	N=0跳转
BRK	Force Break	软件中断	强迫中断
BVC	Branch on Overflow Clear	溢出位为零跳转	V=0跳转
BVS	Branch on Overflow Set	溢出位为1跳转	V=1跳转
CLC	Clear Carry Flag	进位位清零	0→C

续表 1

记忆符	英　　文	中　　文	操　　作
CLD	Clear Decimal Mode	十进制方式位清零	0→D
CLI	Clear Interrupt Disable Bit	禁止中断清零	0→I
CLV	Clear Overflow Flag	溢出位清零	0→V
CMP	Compare Memory and Accumulator	比较	A - M
CPX	Compare Memory and Index X	X比较	X - M
CPY	Compare Memory and Index Y	Y比较	Y - M
DEC	Decrement Memory by One	减 1	M - 1→M
DEX	Decrement Index X by One	X减 1	X - 1→X
DEY	Decrement Index Y by One	Y减 1	Y - 1→Y
EOR	"Exclusive-Or" Memory with Accumulator	逻辑异或	A⊻M→A
INC	Increment Memory by One	加 1	M + 1→M
INX	Increment Index X by One	X加 1	X + 1→X
INY	Increment Index Y by One	Y加 1	Y + 1→Y
JMP	Jump to New Location	无条件跳转	跳转到新地址
JSR	Jump to New Location Saving Return Address	转子	跳转到子程序

续表 2

记忆符	英　文	中　文	操　作
LDA	Load Accumulator with Memory	取数送A	M→A
LDX	Load Index X with Memory	取数送X	M→X
LDY	Load Index Y with Memory	取数送Y	M→Y
LSR	Shift Right one Bit (Memory or Accumulator)	右移一位	0→[7 0]→C
NOP	No Operation	空操作	不进行操作
ORA	"OR" Memory with Accumulator	逻辑或	A∨M→A
PHA	Push Accumulator on Stack	A进栈	$A \to M_S \quad S-1 \to S$
PHP	Push Processor Status on Stack	P进栈	$P \to M_S \quad S-1 \to S$
PLA	Pull Accumulator from Stack	A出栈	$S+1 \to S \quad M_S \to A$
PLP	Pull Processor Status from Stack	P出栈	$S+1 \to S \quad M_S \to P$
ROL	Rotate One Bit Left (Memory or Accumulator)	循环左移一位	←[7 0]←[C]←
ROR	Rotate One Bit Right (Memory or Accumulator)	循环右移一位	→[C]→[7 0]→
RTI	Return from Interrupt	中断返回	中断返回
RTS	Return from Subroutine	子程序返回	子程序返回
SBC	Subtract Memory from Accumulator with Borrow	带进位减法	$A-M-\overline{C} \to A$

续表 3

记忆符	英　　文	中　　文	操　　作
SEC	Set Carry Flag	进位置 1	1→C
SED	Set Decimal Mode	十进制方式位置 1	1→D
SEI	Set Interrupt disable Status	禁止中断位置 1	1→I
STA	Store Accumulator in Memory	A送存	A→M
STX	Store Index X in Memory	X送存	X→M
STY	Store Index Y in Memory	Y送存	Y→M
TAX	Transfer Accumulator to Index X	A送X	A→X
TAY	Transfer Accumulator to Index Y	A送Y	A→Y
TSX	Transfer Stack Pointer to Index X	S送X	S→X
TXA	Transfer Index X to Accumulator	X送A	X→A
TXS	Transfer Index X to Stack Pointer	X送S	X→S
TYA	Transfer Index Y to Accumulator	Y送A	Y→A

对上表中的一些操作符号说明如下：

X　变址寄存器X

Y　变址寄存器Y

A　累加器A

M　有关地址的存贮单元

Ms　堆栈指针所指向的存贮单元

→　传送

+　加号

-　减号

∨　逻辑或

∧　逻辑与

⊻　逻辑异或

表2:　　　　6502　指

INSTRUCTIONS		IMMEOIATE			ABSOLIITE			ZEROPAGE			ACCUM			IMPLIED			(IND,X)		
MNEMONIC	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#
ADC	A+M+C→A (4)(1)	69	2	2	6D	4	3	65	3	2							61	6	2
AND	A∧M→A (1)	29	2	2	2D	4	3	25	3	2							21	6	2
ASL	C←[7 0]←0				0E	6	3	06	5	2	OA	2	1						
BCC	BRANCHONC=0 (2)																		
BCS	BRANCHONC=1 (2)																		
BEQ	BRANCHONZ=1 (2)																		
BIT	A∧M				2C	4	3	24	3	2									
BMI	BRANCHONN=1 (2)																		
BNE	BRANCHONZ=0 (2)																		
BPL	BRANCHONN=0 (2)																		
BRK	BREAK													00	7	1			
BVC	BRANCHONV=0 (2)																		
BVS	BRANCHONV=1 (2)																		
CLC	0→C													18	2	1			
CLD	0→D													D8	2	1			
CLI	0→1													58	2	1			
CLV	0→V													B8	2	1			
CMP	A-M	C9	2	2	CD	4	3	C5	3	2							C1	6	2
CPX	X-M	E0	2	2	EC	4	3	E4	3	2									
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2									

令 表

(IND),Y			Z,PAGE,X			ABS,X			ABS,Y			RELATIVE			INDIRECT			Z,PAGE,Y			PROCESSORSTATUS CODES	
OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	7 6 5 4 3 2 1 0 / N V • B D I Z C	MNEMONIC
71	5	2	75	4	2	7D	4	3	79	4	3										N V • • • • Z C	ADC
31	5	2	35	4	2	3D	4	3	39	4	3										N • • • • • Z •	AND
			16	6	2	1E	7	3													N • • • • • Z C	ASL
												90	2	2							• • • • • • • •	BCC
												B0	2	2							• • • • • • • •	BCS
												F0	2	2							• • • • • • • •	BEQ
																					M_7 M_6 • • • • Z •	BIT
												30	2	2							• • • • • • • •	BMI
												D0	2	2							• • • • • • • •	BNE
												10	2	2							• • • • • • • •	BPL
																					• • • 1 • 1 • •	BRK
												50	2	2							• • • • • • • •	BVC
												70	2	2							• • • • • • • •	BVS
																					• • • • • • • 0	CLC
																					• • • • 0 • • •	CLD
																					• • • • • 0 • •	CLI
																					• 0 • • • • • •	CLV
D1	5	2	D5	4	2	DD	4	3	D9	4	3										N • • • • • Z C	CMP
																					N • • • • • Z C	CPX
																					N • • • • • Z C	CPY

续表 1

INSTRUCTIONS		IMMEDIATE			ABSOLUTE			ZEROPAGE			ACCUM			IMPLIED			(IND,X)		
MNEMONIC	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#
DEC	M−1→M				CE	6	3	C6	5	2									
DEX	X−1→X													CA	2	1			
DEY	Y−1→Y													88	2	1			
EOR	$A \veebar M$→A (1)	49	2	2	4D	4	3	45	3	2							41	6	2
INC	M+1→M				EE	6	3	E6	5	2									
INX	X+1→X													E8	2	1			
INY	Y+1→Y													C8	2	1			
JMP	JUMPTONEWLOC				4C	3	3												
JSR	JUMPSUB				20	6	3												
LDA	M→A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2
LDX	M→X (1)	A2	2	2	AE	4	3	A6	3	2									
LDY	M→Y (1)	A0	2	2	AC	4	3	A4	3	2									
LSR	0→[7 0]→C				4E	6	3	46	5	2	4A	2	1						
NOP	NOOPERATION													EA	2	1			
ORA	A∨M→A	09	2	2	0D	4	3	05	3	2							01	6	2
PHA	A→Ms S−1→S													48	3	1			
PHP	P→Ms S−1→S													08	3	1			
PLA	S+1→S Ms→A													68	4	1			
PLP	S+1→S Ms→P													28	4	1			
ROL	←[7 0]←[C]←				2E	6	3	26	5	2	2A	2	1						

(IND),Y			Z,PAGE,X			ABS,X			ABS,Y			RELATIVE			INDIRECT			Z,PAGE;Y			PROCESSORSTATUS CODES	
OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	7 6 5 4 3 2 1 0 NV • B D 1 Z C	MNEMONIC
			D6	6	2	DE	7	3													N • • • • • Z •	DEC
																					N • • • • • Z •	DEX
																					N • • • • • Z •	DEY
51	5	2	55	4	2	5D	4	3	59	4	3										N • • • • • Z •	EOR
			F6	6	2	FE	7	3													N • • • • • Z •	INC
																					N • • • • • Z •	INX
																					N • • • • • Z •	INY
															6C	5	3				• • • • • • • •	JMP
																					• • • • • • • •	JSR
B1	5	2	B5	4	2	BD	4	3	B9	4	3										N • • • • • Z •	LDA
									BE	4	3							B6	4	2	N • • • • • Z •	LDX
			B4	4	2	BC	4	3													N • • • • • Z •	LDY
			56	6	2	5E	7	3													0 • • • • • Z C	LSR
																					• • • • • • • •	NOP
11	5	2	15	4	2	1D	4	3	19	4	3										N • • • • • Z •	ORA
																					• • • • • • • •	PHA
																					• • • • • • • •	PHP
																					N • • • • • Z •	PLA
																					(RESTORED)	PLP
			36	6	2	3E	7	3													N • • • • • Z C	ROL

续表 2

INSTRUCTIONS		IMMEOIATE			ABSOLIITE			ZEROPAGE			ACCUM			IMPLIED			(IND.X)		
MNEMONIC	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#
ROR	→[C]→[7 0]→				6E	6	3	66	5	2	6A	2	1						
RTI	RTRNINT													40	6	1			
RTS	RTRNSUB													60	6	1			
SBC	$A - M - \overline{C} \to A$ (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2
SEC	1→C													38	2	1			
SED	1→D													F8	2	1			
SEI	1→1													78	2	1			
STA	A→M				8D	4	3	85	3	2							81	6	2
STX	X→M				8E	4	3	86	3	2									
STY	Y→M				8C	4	3	84	3	2									
TAX	A→X													AA	2	1			
TAY	A→Y													A8	2	1			
TSX	S→X													BA	2	1			
TXA	X→A													8A	2	1			
TXS	X→S													9A	2	1			
TYA	Y→A													98	2	1			

(IND),Y			Z;PAGE;X			ABS,X			ABS,Y			RELATIVE			INDIRECT			Z,PAGE, Y			PROCESSORSTATUS CODES 7 6 5 4 3 2 1 0	MNEMONIC
OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	N V • B D I Z C	
			76	6	2	7E	7	3													N • • • • • Z C	ROR
																					(RESTORED)	RTI
																					• • • • • • • •	RTS
F1	5	2	F5	4	2	FD	4	3	F9	4	3										NV • • • • Z (3)	SBC
																					• • • • • • • 1	SEC
																					• • • • 1 • • •	SED
																					• • • • • 1 • •	SEI
91	6	2	95	4	2	9D	5	3	99	5	3										• • • • • • • •	STA
																		96	4	2	• • • • • • • •	STX
			94	4	2																• • • • • • • •	STY
																					N • • • • • Z •	TAX
																					N • • • • • Z •	TAY
																					N • • • • • Z •	TSX
																					N • • • • • Z •	TXA
																					• • • • • • • •	TXS
																					N • • • • • Z •	TYA

以上列出的是指令表，对它我们说明如下：

1.表中的注释：

(1) 如果跨页，指令执行时钟周期数(n)加1。

(2) 如果有同一页内跳转，指令执行时钟周期数(n)应加1；若跨页跳转指令执行时钟周期数(n)则加2。

(3)进位非＝借位　($\overline{C}$＝借位)

(4)在十进制方式中，Z标志位无效。结果为0必须检查累加器。

2. 表中的符号：(上面已介绍的不再重复)

OP　　操作码

#　　字节数

n　　指令执行时钟周期数

M_7　　存贮单元的*Bit*7

M_6　　存贮单元的*Bit*6

MNEMONIC　　记忆符

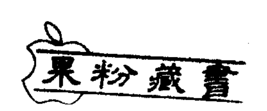
INSTRUCTION　　指令

PROCESSOR STATUS CODES　　标志寄存器标志码(处理机状态码)

3. 整个表大致可以分为五栏。从左到右第一栏、第五栏是记忆符。第二栏是说明每条指令所执行的操作(*OPERATION*)。第三栏是本表的主要部分，它并排列出了十三种寻址方式。以下是各种寻址方式所对应的指令的操作码(OP)、执行时钟周期数(n)和字节数(#)。操作码是用十六进制数表示的。第四栏表示的是每条相应的指令执行以后对标志寄存器(或称处理机状态寄存器)各位的影响。其中“·”表示没有影响，1表示置1，0表示置0。“N”、“Z”、“C”、“V”表示要根据指令执行的结果而确定究竟是0还是1。“RESTORED”表示恢复原来所保存的值。“M_7”、“M_6”请见BIT指令的说明。

4．有关十三种寻址方式说明如下：

IMMEDIATE	立即寻址（直接寻址）
ASSOLUTE	绝对寻址
ZERO PAGE	零页寻址
ACCUM.	累加器寻址
IMPLIED	隐含寻址
(IND, X)	先变址（X)的间接寻址
(IND), Y	后变址（Y)的间接寻址
Z.PAGE, X	X零页变址寻址
ABS, X	X绝对变址寻址
ABS, Y	Y绝对变址寻址
RELATIVE	相对寻址
INDIRECT	间接寻址
Z.PAGE, Y	Y零页变址寻址



6502微处理机及其应用

荣树熙　张开敬 编

*

北京师范大学出版社出版

新华书店北京发行所发行

湖南省新华印刷二厂印刷

*

开本：850×1168 1/32　印张：12.75　字数：310千

1984年12月第1版　1984年12月第1次印刷

印数：1—42,000

统一书号：13243·66　　定价：3.10元

封面设计：金国辉

科目：85—75
统一书号：13243·66
定 价：3.10 元